

Introduction to ASCI Q



Harvey Wasserman

Los Alamos National Laboratory
HPC Systems Group (CCN-7)

May, 2006

ASCI Training
@LANL
<http://asci-training.lanl.gov>



**TABLE OF CONTENTS**

Beginning-Level Topics

Course Objectives**Cluster Systems at LANL****Q Hardware****Logging In**

- Front End Names
- Front End Logins
- Back End Logins

General Computing

- Operating System
- Moving Files
- 3rd Party SW
- Miscellaneous

Modules**Storage****Filesystems**

- Home Directories
- Scratch Directories
- Where Should I Work?

LSF on Q

- Q Resource Management
- Q Cluster Domains
- Submitting Jobs
- LSF Resources on Q
- RMS on Q
- Modifying LSF Jobs
- Error Messages
- LSF Scripts
- LSF Queue Structure

Compiling

- Compiling MPI codes

OpenMP on Q

Introduction to ASCI Q - Beginning

Overview and Course Objectives

- This is a course designed to introduce users to the ASCI Q systems at LANL. A prerequisite is an understanding of LSF as it is used at LANL. Information on LSF is in a separate [LSF tutorial](http://ascii-training.lanl.gov/LSF), <http://ascii-training.lanl.gov/LSF>.
- The course contains information for both "users" and developers and the content is "front-loaded" so that most of the "user" information is in the beginning and most of the developer information is towards the end.
- You can use these materials to study on your own but please keep in mind that they are intended to accompany a one-day classroom course.

Contents

Cluster Computing Systems at LANL

- LANL's high-performance computing resources now consist almost entirely of **clusters**.

What is a Cluster?

A cluster is a collection of computing resources - servers, storage, and interconnects - that can function as a single system in terms of management, access, and applications.

Cluster computing is used for load balancing, for providing high availability, and for improving cost-performance for parallel computing.

- Important cluster systems at LANL today are the machines known as:

- | | |
|----------------------------|-----------------------------|
| • QA | • QB |
| • CA | • CB |
| • CC | • CX |
| • QSC | • Lambda |
| • Grendels | • Flash |
| • TLC | • Lightning |
| • Pink | |

This course concerns those systems marked with the symbol.

Contents

ASCI Q Clusters at LANL

- Each Q cluster is comprised of some number of "nodes" (aka "back ends," "compute servers") and an interconnect.
- In all cases, the node is an HP Alphaserver SC, also known as an ES45.

Debugging on Q

Where to Go for Help

Reference Info

Advanced-Level Topics

More LSF Topics

- Reservation & Backfill
- Fairshare

Architecture

- General
- Alpha Processor
- Quadrics
- Storage

I/O on Q

- File management
- Small Operations
- Large Serial Files
- Parallel N to N
- Parallel N to 1
- Parallel Copies
- HPSS Performance

Performance Analysis

- Timing a Code
- gprof
- Pixie
- HiProf
- uprofile
- Misc. Tools

Course Evaluation

- In all cases, the node contains 4 Alpha EV68 1.25-GHz processors. Peak performance is 10 GFlops per node.
- In all cases, the interconnect is the Quadrics QsNet.
- In all cases, the operating system is HP Tru64



QA and QB:

- Each of these is composed of 1,024 nodes
- 64 nodes @32 GB memory per node; 192 @ 16 GB; 768 @ 8 GB
- 40 TB of RAM total
- 10 TF peak performance
- Dual-Rail Quadrics Interconnect
- QA and QB can be combined (in a few hours) creating a single system
- Used primarily for capability computing.
- QA Available to the Tri-Lab ASCI community.
- Classified systems.



CA, CB, and CC

- Each of these is composed of 128 nodes
- (LANL) Terminology: We refer to this as a cluster containing three 128-node "segments."
- 4 GB memory per node
- 1.3 TF peak performance per segment
- Single-Rail Quadrics Interconnect
- Will not be combined to create a single system
- Used entirely for capacity computing - DSW jobs requiring relatively few processors.
- Classified systems



CX

- 32 nodes
- 2 nodes @ 16 GB memory per node; 30 nodes @ 4 GB memory per node
- Single-Rail Quadrics Interconnect
- No parallel file system
- Classified system for legacy code users (replaced machine QX).



QSC

- 256 nodes
- 16 GB memory per node
- Dual-Rail Quadrics Interconnect
- Unclassified system, for Institutional Computing.

And just for comparison:



Blue Mountain

- 48 nodes
- 128 processors per node
- 16-32 GB memory per node
- HiPPI or GSN interconnect



Comparison of LANL Computing Systems	
Blue Mountain	QA
SGI Origin2000	HP AlphaServer ES45
250 MHz (4.0-ns clock period)	1,250 MHz (0.8-ns clock period)
128 processors per node	4 processors per node
500 MFLOPS peak per processor	2,500 MFLOPS peak per processor
64 GFLOPS peak per 128-processor SMP	10 GFLOPS peak per 4-processor SMP
3.1 TFLOPS peak per the whole system	10 TFLOPS peak per the whole system
64 GB memory (512 MB per processor)	8-32 GB memory (2-8 GB per processor)
Irix operating system	Tru64 operating system
"superscalar" microprocessors / CMOS technology	"superscalar" microprocessors / CMOS technology

- Example comparison of Q and BlueMountain processors: Schulz hydrodynamics benchmark, 300 x 300 grid for 100 timesteps runs in about 135 seconds on Theta, about 28 seconds on CX (~5 X speedup) **YMMV!**
- This is a "birds-eye" view of the hardware. More detailed description, including explanation of such terms as "dual-rail," "superscalar," and "UMA," may be found in the [Architecture](#) section.

[Contents](#)

Logging In

Logging in to the Front End

- As with most other LANL clusters, there is a front-end/back-end system for all Q clusters. Use [ssh](#) to reach the front end; See list of front end names in the table below. Authenticate with LANL tokencard.
- Computing is not done on the front ends, which exist primarily to control access to the back-end systems.

Q System Front Ends		
System	Front End Names	
Secure	QA	qafe (qfe1)
	QB	qbfe (qfe2)
	CA	cfe1 and cfe2
	CB	
	CC	
	CX	cxfe
Open	QSC	qscfe1 and qscfe2

Logging in to the Back End

- From the front-end, issue an [llogin](#) command to start an interactive session on one of the AlphaServerSC "back-end" compute nodes.
- This command, which has several options that we'll discuss later, submits a request to the resource management system to allow you to have an interactive [tcsh](#) session on a Q back end. If successful, you will be logged into a back end server host.
- Some of the Q systems limit the number of [llogin](#) interactive sessions a user can have at one time.

However, you can start multiple terminal sessions connecting your desktop to the back end(s) by issuing the Unix [xterm](#) command. If you have a kerberos ticket on the front-end and issue an [llogin](#) command to reach a back end, in most cases the X-windows application running on your local machine will successfully display back to the local system. You may have to use the [-x](#) option to [ssh](#) and you may have to use the [xhost](#) command on your local workstation to allow it to accept connections from the remote back-end system.

Note: Recent versions of desktop SSH software sometimes require [-Y](#) instead of [-X](#).

- After some period of time you will be logged out and placed back on the front end. [Here's](#) what that looks like, just in case you're curious. Or, you can type ["exit"](#) to log out.

Exercise

Practice Logging In

- Make sure that you can log in to a Q system from the classroom workstation or from some other LANL host. Use **ssh**.
- Type "**pinfo -v**" on the front end. This command tells you about the processors on the machine. You should see that the front end has only two processors, running at 667 MHz.
- Now log in to the back end with the command that we learned for this.
- Type "**pinfo -v**" again. Note the difference from above.
- See if you can start an xterm on the back end.
- Kill your X session and then **log out** of the back end.

This ends the first exercise.

Contents

General Computing

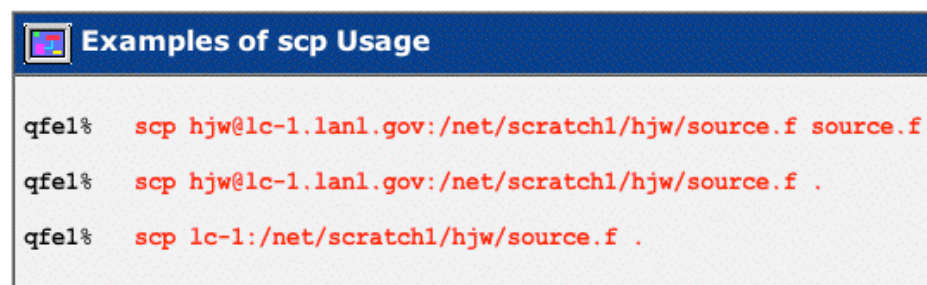
Operating System

- All of the Q clusters run an HP/Compaq product called AlphaServer SC, which is a software package containing the Compaq Tru64 UNIX V5.1b system, a 64-bit OS based on BSD, Mach, System V, and OSF, and various other packages, such as resource management software, the compilers, Pthreads, performance tools, math libraries, debugging systems, various drivers and low-level libraries for the Quadrics interconnect, and OpenMP.
- Main documentation for this software product is available [here](#) (yellow partition only).
- To the user, Tru64 should look basically like any other Unix.
- All LANL systems now run the same OS version, known precisely as "TruCluster Server V5.1-SC-V2.6-UK1-5250 (Rev. 15250)"
- The OS typically uses about 1GB of memory on most nodes and as much as 5 or so on some others. Use the **vmstat -P** command to determine this. The "free pages" line is the number of 8KB pages available in the system. Multiply by 8 and you have the free space in KB.

Moving Files

- The secure copy command, **scp**, is the best method for moving files between machines. The full syntax for this command is:

```
scp [[user@]host1:]filename1... [[user@]host2:]filename2
```



- Caution: What's wrong with this one? What will be the result of this command?

```
trinidad% scp hydro.f hjw@cxfe
```

Answer is [here](#)

- Also available are a variety of utilities, such as **ftp**, **sftp**, **k5ftp**, and **k5rcp**. Authenticate with your secure ICN CryptoCard. Note: certain directories on the Q systems are not accessible via **scp** or any of the ftp commands. See the "Filesystems" section below.
- The **give** utility also works on the Q machines. It sends a copy of the named local file to a temporary system directory for the intended user to copy.

To use it, type:

```
give filename userid
```



Note: Same syntax as on Blue Mountain but different destination directory.

- A directory for the recipient is created on **/givedir/userid** and the file is put in there. The recipient must then copy the file from the **/givedir/userid/** location.



A single **/givedir** is "global" across CX, CA, CB, and CC but **not** QA/QB. Both QA and QB share a **/givedir** but this is not shared with the Cs.



As of February, 2004 there is a purge policy in place for **/givedir** on all of the HP systems. It runs on QA, QB, C's, and QSC. It runs once every hour and deletes files not accessed in the last **4** days. This is because when this filesystem is full, and it does occasionally fill up, the give utility will not work. **Not the same as on theta, BM, or lambda.** Don't use this space for storage.

- You can use the **give** command on the front ends and you can access the destination directories on the front ends.
- Note for LLNL users: There is no **take** command. Just go to the destination directory and copy the file.
- Note: A nice alternative to **give** is the new give command within PSI v5. Usage:

```
psi give [-d dir] [-Q] fileList usr
```

See the [New PSI man page](http://int.lanl.gov/hpss/PSI.html) at <http://int.lanl.gov/hpss/PSI.html> for more info.

Miscellaneous Software Issues

- Default login shell on all LANL ASCII systems is **tcsh**. HP (the machine's manufacturer) does not support **bash** as a default login shell. If you insist on using it, change your shell after you log in. Shells other than **tcsh** are basically at the "use at your own risk" level.
- Important consideration: if you are not a code developer and/or don't expect to be debugging, then set your environment up so as not to dump core:

```
limit coredumpsize 0
```

This will prevent the creation of core files entirely.

Another way to do this on the Q systems is with shell variables:

In csh and tcsh use **setenv RMS_KEEP_CORE FALSE**

In sh and bash use **export RMS_KEEP_CORE=FALSE**

- On Q systems a core file is produced (assuming you don't have the above options set) in the directory **/local/core/rms**, which will contain a subdirectory for you labeled with a number. (This number is assigned by the resource management system and is a unique identifier for your LSF session.) Run **ls -al** in this directory to see which subdirectory is yours. Then **cd** to that directory. Core files are of the format

```
core.<executable_name>.<machine_name>.<thread_#>.
```

Exercise

Find Your Core

1. From the last exercise you should be logged in to a Q system back end. Verify.
2. Run the following program (just type its entire name):
/scratch1/hjw/nan .
3. You should see that this program crashes. Find the core file that was produced.
4. If no core file was produced then reset your system limits (type **unlimit**), then run the program again and find the core file.
5. Then log out of the back end.

This ends the core exercise.

Contents

Modules

- At LANL all important system software (such as compilers, tools, and libraries) can be accessed **ONLY** within programming environment packages called **modulefiles**. *You do not have access to these by default.*
- The Modules package has two main purposes:
 - It provides a convenient way to make multiple versions of system software available to users.
 - It provides a convenient way for users to choose a particular version of the software and update their working environment.
- The Modules package consists of two parts: a shell-level **module** utility/command and the **modulefiles** that this utility uses.
- You need to learn about (and use) the Module package regardless of whether you're a code developer or code user.
- Modulefile names are case-sensitive.
- To initialize the modules environment, your **.cshrc** file should contain the following:

```
if (-f /opt/modules/modules/init/tcsh) then
    source /opt/modules/modules/init/tcsh
endif
if (-f /opt/modules/modules/modulefiles/modules) then
    module load modules
endif
```

Note: Do not use this code on a LANL BProc system.

- It's a good idea to include the above code, and the corresponding code for any modules that you need to load, in any script files that you use on Q machines. Many users use scripts that routinely load and list modulefiles just prior to running an application. Use caution if you write scripts using a shell that's different from your login shell - you have to initialize the Modules package for the shell you're going to use.



All the module-related software exists **ONLY** on the back ends on Q.



Error messages associated with module-related problems can be obscure. Two examples are shown below.

In the first case, a user is attempting to run an MPI-based executable without having loaded an MPI modulefile. In the second, we see that compilation of MPI codes cannot be done without first loading a modulefile. (However, we will see later that loading the modulefile is not sufficient for MPI compiles.)

```

q123% brun -n 4 ./sweep3d.mpi
sweep3d.mpi: /sbin/loader: Fatal Error: Cannot map library libfmpi.so
sweep3d.mpi: /sbin/loader: Fatal Error: Cannot map library libfmpi.so
sweep3d.mpi: /sbin/loader: Fatal Error: Cannot map library libfmpi.so
sweep3d.mpi: /sbin/loader: Fatal Error: Cannot map library libfmpi.so
  
```

Annotations:

- Shell prompt: points to `q123%`
- User command: points to `brun -n 4 ./sweep3d.mpi`
- Strange error message: points to the fatal error text.

```

q123% f90 -fast -c mpi_stuff.f
f90: Error: mpi_stuff.f, line 16: Cannot open include file 'mpif.h'
include 'mpif.h'
-----^
f90: Error: mpi_stuff.f, line 42: This name does not have a type,
and must have an explicit type.      [MPI_COMM_WORLD]
call mpi_comm_size(MPI_COMM_WORLD, size, info)
-----^
  
```

- Take a look [here](#) to see the modulefiles that were available on QSC one random day during the winter of 2004:

Module commands

module avail [package]

This lists all the modulefiles that are available. Notice that many of them have version numbers associated with them. You can see which modulefiles are considered to be the production default.

module load modulefile

This adds one or more modulefiles to your current environment. It does so silently, unless there is a problem with a modulefile. There are no "generic" names of a module; you must load exactly the version that you want.

module list

This lists all the modules which are currently loaded into your environment.

module unload modulefile

This removes any [listed](#) modulefiles from your current environment. The modules can be removed in any order.

module switch modulefile_old modulefile_new

This command demonstrates the true advantages of modules. Different versions of entire software packages can be replaced with a single module command.

module purge

This command takes no options and will unload all of your loaded modulefiles. Use it with care, because it will also unload the module modulefile itself!

module display modulefile

Use this command to see exactly what a given modulefile will do to your environment, such as what will be added to the **PATH**, **MANPATH**, etc. environment variables. Same as **module show modulefile**

module help

This lists all the options available but for real help, consult the [man](#) pages.



Finally, the module utility generally doesn't check for potential conflicts between modulefiles on the Tru64 Q systems. Avoid loading two modulefiles of the same software product. Otherwise, you may see very strange errors that are difficult to diagnose. Before loading a second modulefile of the same name (different version), be sure to first unload the first one, or use the **module switch** or **module swap** commands.

The module command produces its terminal output on *stderr* (not *stdout*), so if you want to, for example, redirect output to the file "out" you have to use (from csh):
module avail > & out.

Contents

Archival Storage

- HPSS is available on the Q systems. You can access HPSS via:
 - [psi](#). This is a LANL-grown command interface with Unix-like syntax.
 - In mid- or late-2005, the version of psi on all Q systems changed from v3 to v5. Version 5 offers performance improvements and many new features. See <http://int.lanl.gov/hpss/NewPSI.php> for more information.
 - In order to get the most benefit from psi v5 you need to use multiple Q nodes. Do this especially when transferring large files and/or lots of files.

At LANL there is a tutorial on psi available as part of the "Intro to ASCI Blue Mountain" tutorial on <http://asci-training.lanl.gov/>
 - pftp (pftp@hpss.lanl.gov). You shouldn't have to authenticate but its performance for large files and for large file trees will generally not be as good as what you'll get from psi.
- For more info on HPSS visit the web page: <http://int.lanl.gov/hpss>
- There will be some info on how you might improve HPSS performance later in this class.

Contents

Filesystems

Home Directories

- Each of the ASCI Q systems provides access to your home directory, which is maintained in a central Network File System ([NFS](#)) server. and is therefore, identical to your home directory on other systems:
 - QA, QB, CA, CB, CC, CX, Lightning, and Blue Mountain all the same in the red partition.
 - QSC, Flash, pfe2, Lambda, and Theta all the same in the yellow partition.
- Your home directory will be in one of the four locations
[/users1/user_id](#), [/users2/user_id](#), [/users3/user_id](#), [/users4/user_id](#).

NOTE: This is different than it was on the SGI systems (Blue Mountain and Theta).

You should always reference your home directory in scripts, login files, and/or programs using the `$HOME` variable or via `~user_id` and not with the absolute path `/usersN` because we may move your home directory from one NFS server to another without telling you.

- The disk quota for home directories is 5 GB in the secure and 3 GB in the open. Use the command `quota -a` to get details. (In other words, `quota` by itself doesn't work. See the [ICN Consultants' Tips](#) for more information on this.)
 - Caution: A quota-exceeded condition could mean that you can't log in to the back ends with LSF. An error message in such a case might look like [this](#).
- Home directories have a temporary backup/archival [scheme](#) within the `.snapshot` directory.
 - This is a read-only, hidden directory (not shown with `ls` but you can `cd` into it).
 - Contains multiple subdirectories called `hourly.#` and `nightly.#` each of which contains a full backup from a recent period of time (hourly.0 is most recent).
 - Note that these directories won't get you through a weekend!.

"Project" Directories

- Primarily a place for code, input decks, executables, scripts, data files, etc., to be used by large group of people and potentially shared across several different LANL clusters, e.g., QA/QB/Lightning/C.
- Also backed up via a temporary backup/archival scheme within the `/usr/projects/.snapshot` directory.
- Two ways to obtain this space:
 1. If you are a project leader and your project has a unix group established already on register.lanl.gov send an e-mail to consult@lanl.gov requesting the space. Include a justification and an estimate of space required. However, your request may be denied and redirected instead to option ...
 2. Establish it yourself by creating a new directory in `/usr/projects/packages`. Any user can do this - The `/usr/projects/packages` space is world r/w-able. .

Temporary Storage

Q Clusters Temporary File Systems				
	Cluster	Scratch Directories	Capacity	Notes
Secure	QA	/scratch1 /scratch2	40TB	QA can't see QB's scratch space; Front end can't see scratch space
	QB	/scratch1 /scratch2	40TB	QB can't see QA's scratch space; Front end can't see scratch space
	CA	/scratch1	9 TB	CA can't see QA's, CB's, CC's, or CX's scratch space; Front end can't see scratch space
	CB	/scratch1	9 TB	Etc.
	CC	/scratch1	9 TB	Etc.Etc.
	CX	/scratch	5 TB	The CX /scratch temporary file space is optimized for serial access and is visible from cxfe
	All Secure Systems (QA, QB, CX, CA, CB, & CC)	/netscratch	30TB	A new NFS filesystem with 100GB for each user id. Available on front ends and back ends.
Open	QSC	/scratch1 /scratch2	12 TB	Front end can't see scratch space

- Note that some of these scratch filesystems can only be "seen" from the back ends. Example:



Scratch Directories Do Not Exist On the Front Ends

```
qafe% cd /scratch1/hjw
/scratch1: No such file or directory.
```

- The **/netscratch** filesystems are global across all of the secure systems (same directory for a given userid on BlueMountain, QA, CX, etc. etc.).
- There is no purge policy on the **/netscratch** filesystems.
- Avoid having multiple soft links between files in the various Q filesystems, for example, between a file in your home space and a file in **/netscratch**.
- The **/netscratch** filesystems are also served by four different servers (**netscratch5**, **netscratch6**, **netscratch7**, and **netscratch8**). This is what you see when you use the **quota -a** command to check your usage. Only one of them will show usage for you.



Example quota -a Output

```
qfel% quota -a
Disk quotas for user hjw (uid 1379):
```

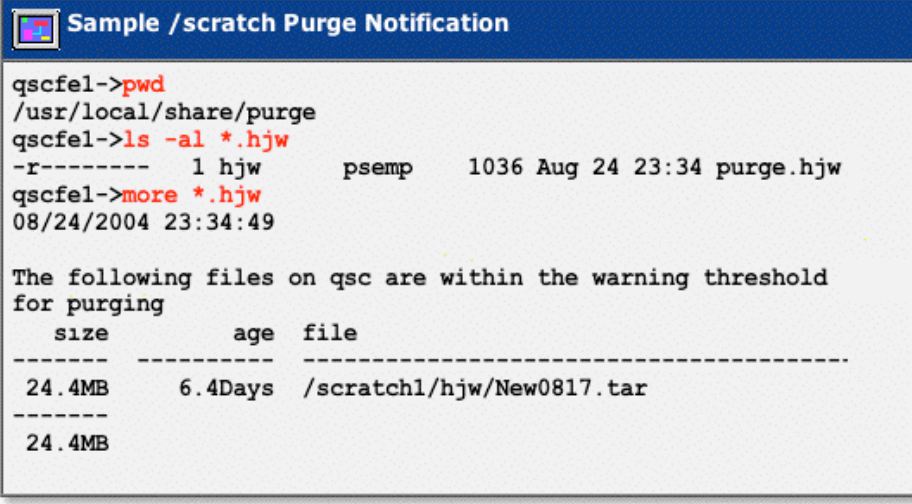
Filesystem	blocks	quota	limit	grace	files	quota	limit	grace
/netscratch5	0	52428800	5248800	0	204800	204800		
/netscratch6	0	52428800	5248800	0	204800	204800		
/netscratch7	0	52428800	5248800	0	204800	204800		
/netscratch8	8748	52428800	5248800	0	204800	204800		
/users1	86345	5242880	5242880					
/users2	0	5242880	5242880					
/users3	0	5242880	5242880					
/users4	0	5242880	5242880					

- The **/scratchN** filesystems of the various Q clusters are mutually exclusive. They are not shared or cross-mounted.
 - A file in **/scratch1** or **/scratch2** on QB may not be accessed from QA and *vice versa*.
 - To transfer files between two **/scratch1** filesystems on different clusters (including different Q clusters and/or BlueMountain), you need to use either the special file transfer routines explained below or use (HPSS) as an intermediary.
- There are no user quotas for the **/scratch** filesystems. However, space is very limited; therefore, ...
- **Purge Policy:** The following applies to the **/scratch** filesystems on **all** of the LANL Q clusters:
 - ALL FILES now eligible for purging.
 - Files larger than 5 MB will be purged if last access is more than 7 days. Users will be notified by e-mail 4 days before files are purged.
 - Files smaller than 5 MB will be purged if last access is more than 16 days; notification by e-mail 8 days before purging.
 - Purging will "24 x 7" - meaning even weekends, holidays, and during Lab closures. However, purging will occur only on machines that have 50% or less of **/scratch** space available.
 - This rather drastic purge policy is in place because these file systems are continually filling up. When they get close to the limit, there is a real possibility of losing all data in the file system. Hence it is to everyone's advantage for you to delete all files that are not actually being used on a daily basis.
 - **Important caution:** The purger looks at the file modification/create date. Therefore, just using (reading) a file will not "touch" it and stop the purger.

- The directory `/usr/local/share/purge` will contain a file of the form `purge.<moniker>` or `purge.<moniker>.<mx>`, where `<moniker>` is your user id and `<mx>` is one of the Q systems (qa, qb, ca, cb, cc, or cx) that contains info relating to your purge situation. The directory is global across all the Q systems and can be viewed from the front end.
- There's a script that sends you e-mail regarding upcoming purges of your files. If you don't want to get those e-mails you can use the following command:

```
touch ~/.nopurgemail
```

However, if you do this you also won't have a purge file in `/usr/local/share/purge`.



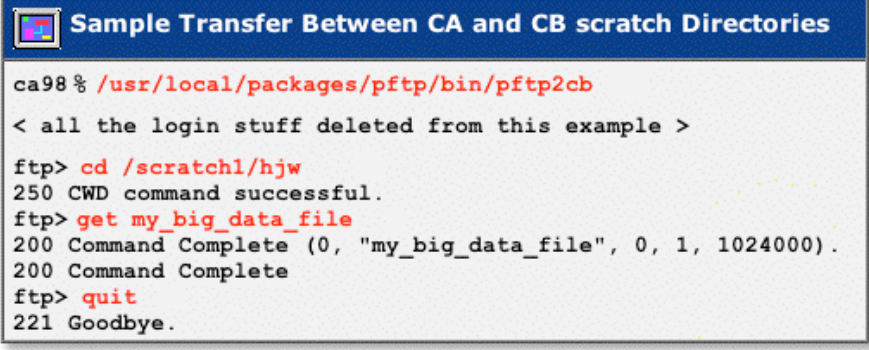
Sample /scratch Purge Notification

```
qscfel->pwd
/usr/local/share/purge
qscfel->ls -al *.hjl
-r----- 1 hjl      pscmp    1036 Aug 24 23:34 purge.hjl
qscfel->more *.hjl
08/24/2004 23:34:49

The following files on qsc are within the warning threshold
for purging
  size      age      file
-----
 24.4MB     6.4Days  /scratch1/hjl/New0817.tar
-----
 24.4MB
```

- On some of the Qs there are special temporary filesystems used by those involved in milepost calculations. If you are one of these users make sure you don't leave files in those locations after your milepost is completed - they will be purged. These `/mileN` filesystems share the same disks as the `/scratchN` filesystems. More on this later.
- When the `/scratch` filesystems start having problems jobs writing to them tend to hang. If you observe this please contact the LANL ICN Consultants (consult@lanl.gov or 505.665.4444 option 3).
- **Very important caution:** Each of the Q systems is removed from service regularly, once a month, to perform system maintenance and diagnostics. (The date is announced both via e-mail to all users and via the web site <http://icnn1.lanl.gov/webcalendar/month.php>.) A key point is that typically, the scratch filesystems remain offline for 1-3 days after the maintenance period is over. This is for file system checks. So make sure your data are backed up!!!
- **Inter-Cluster /scratchN File Transfer.** There are special file transfer utilities available in the secure partition. These are scripts that fire up pftp sessions between wherever you are and certain remote hosts. They are all located in `/usr/local/bin` or `/usr/local/packages/pftp/bin`. They can be used on the front ends.




Some of them are for doing direct inter-Lab transfers but there are also some that you can use to transfer files between the scratch filesystems of different LANL clusters.




Sample Transfer Between CA and CB scratch Directories

```
ca98% /usr/local/packages/pftp/bin/pftp2cb
< all the login stuff deleted from this example >

ftp> cd /scratch1/hjl
250 CWD command successful.
ftp> get my_big_data_file
200 Command Complete (0, "my_big_data_file", 0, 1, 1024000).
200 Command Complete
ftp> quit
221 Goodbye.
```

-  Perhaps the most important thing you should take home from this discussion on scratch filesystems (including netscratch) is that they are not intended to be permanent storage.
-  They are NOT backed up nor protected via .snapshot.
-  Back up your data! When these /scratch file systems fill up, no one can get any work done! Please delete files from the /scratch file systems as soon as you are through with them.

 **It Could Happen To You...**

Subject: Lost Scratch1 on QA
From: consult@lanl.gov
Date: Fri, April 16, 2004 12:59pm
To: q-users@lanl.gov
CC:
Priority: Normal

Data on Scratch1 on QA cannot be repaired.
All Data on QA Scratch1 are lost.
The rebuild of scratch1 is beginning and it
is hoped scratch1 space will be available COB Monday.

Subject: QSC /scratch2 failure
From: "ICN Consultants" <consult@lanl.gov>
Date: Wed, March 29, 2006 4:59 pm
To: ICN-Notifications@lanl.gov
Priority: Normal
Options: [View Full Header](#) | [View Printable Version](#)

To: gsc-users@lanl.gov

The QSC file system /scratch2 has experienced a failure that corrupted the metadata so that all the inode information has been lost.

The system managers will begin rebuilding the file system on Thursday morning.

We apologize for this and hope that your important files have been backed up to HPSS.

ICN Consulting Office
505-665-4444, option 3
consult@lanl.gov

Exercise

Practice Using Modules, tcsh, and the Filesystems

1. See what modulefiles are available on the workshop Q system. Think about which ones you are likely to be using.
2. Note: If, carrying out the previous instruction, you tried a **module** command and got either nothing or an error message, possible reasons for this include:
 - You're still on the front end and we know that the module utility only works on the back end.

- You need to initialize the module environment. Do this by copying the shell code given in the tutorial into your favorite "dot" file. You should be able to copy and paste directly from the browser into your shell.
3. If you still cannot see modules please contact the instructor.
 4. If you find some modulefiles that you will routinely need you might want to edit your favorite "dot" file and add the conditional `module load ...` to load them. This code is of the form:


```
if (-f /opt/modulefiles/fortran_default) then
    module load fortran_default
endif
```
 5. Verify that you are able to use the three temporary file spaces, `/scratch1`, `/scratch2`, and `/netscratch` (unless you're working on QSC, which does not have `/netscratch`, or CX, which has only `/scratch` and `/netscratch`. Check your quota and make sure you understand the output. Remember that you need to use `quota -a` on the Tru64 systems.
 6. Continue here only if you are not familiar with `tcsh`. This exercise will help you become familiar with one of its nicest features - USING THE ARROW KEYS FOR COMMAND LINE EDITING.
 7. Type `"cd /usr/local/package/totalview.6.0.0-1"`
 8. This command will fail. I really wanted you to go to `"/usr/local/packages/totalview.6.0.0-1,"` i.e, it should be `"packages"` not `"package"`. So now:
 - Press the "up" arrow key once to capture the last command
 - Press and hold the "left" arrow key until the cursor is just after the "e" in package
 - then type an "s"
 - then hit return.
 - Then do `"pwd"` just to prove that the editing worked

This ends the exercise on modules and filesystems.

Where Should I Do My Work?

- We have the following 5 kinds of file spaces on the Tru64 Q clusters:
 1. Each node in the Q clusters has a local disk, about 120 GB. This storage is mounted on `/local` or `/tmp`.
 2. NFS-mounted home directories (`$HOME`).
 3. NFS-mounted "project" directories, mounted on `/usr/local/projects/proj_name`.
 4. NFS-mounted `/netscratch` temporary storage
 5. PFS-mounted `/scratch1` and `/scratch2` (and `/mile1`) temporary storage. [PFS](#) stands for Parallel File System; it is HP's proprietary scalable SAN. Some characteristics include:
 - It is composed of many individual components but it can be accessed and viewed as a single, large filesystem.
 - Data written to this filesystem can be striped across several components.
 - It uses a series of "file servers" to service data requests from all nodes in the cluster. Data are transmitted to/from the file servers using the Quadrics interconnect.
- The question we want to answer is, "where should I do my runs?" The answer is simply that there is no one answer; rather, there are a series of tradeoffs involving the following

characteristics:

- I/O performance
- amount of space available
- total amount of data read/written
- I/O access pattern
- visibility across Q clusters
- data availability, which is inversely proportional to
- file system fragility
- front-end accessibility

Here is a summary of all the tradeoffs:

/tmp

- Performance: Excellent - I/O operations are cached in memory.
- Available Space: 120 GB or less, shared with memory & system SW
- Availability /Fragility: Extremely fragile; NO GUARANTEE OF ANY PERSISTENCE; no backup; files in /local are removed at the end of your LSF job! If /local fills up, the node will crash.
- Front End: Not accessible
- Notes: Use for small operations that often can benefit from local caching such as tar, make, (i.e., compiling), and configure.

\$HOME

- Performance: Fair but non-scalable - uses NFS protocols
- Available Space: 5GB; if you exceed quota, you can't log in to the back ends
- Availability /Fragility: Highly available; backup via .snapshot; Safest space
- Front End: Accessible
- Notes: There's no place like \$HOME.

Projects (/usr/projects)

- Performance: Fair but non-scalable - uses NFS protocols
- Available Space: limited
- Availability /Fragility: Highly available; backup via .snapshot; Safest space
- Front End: Accessible
- Notes: Use for code, executables, system-wide input files (such as X-sections, EOS, etc.)

/netscratch

- Performance: Fair but non-scalable - uses NFS protocols
- Available Space: 50-GB hard limit per user
- Availability /Fragility: Highly available but no backup
- Front End: Accessible
- Notes: Best used for serial I/O; also useful for repeated small operations (tar, make); however, make sure you don't exceed your /netscratch quota during a job; if you do, the job will probably hang indefinitely.

/scratch1 and /scratch2

- Performance: Extremely dependent on I/O size, access pattern, and PFS parameters; Small/serial operations can be extremely slow; filestat operations extremely slow with very large numbers of files; large parallel operations run well & scale well
- Available Space: No user quota; filesystem limits and component limits (relevant for jobs doing large single-file serial I/O)
- Availability /Fragility: Fairly fragile; you should back up to archival storage daily; if a PFS filesystem fills, the entire filesystem may crash with loss of user data
- Front End: **Not** Accessible
- Built particularly for large-scale parallel I/O, especially via MPI-IO. Metadata operations can be very slow in the **/scratch1** and **/scratch2** directories. If you wish to delete very large numbers of files in a **/scratchN** directory, a fast method is available. Simply move (**mv**) the files to **/scratchN/trash**. The operating system will purge them during off hours.

- Two additional cautions regarding the Q **/scratchN** parallel filesystems are as follows. These relate to the fact that these filesystems are not fully posix compliant:
 1. Avoid modifying your environment so that it uses a PFS directory as a runtime space; i.e., don't set or append **LD_LIBRARY_PATH** or anything similar so that it has an entry based on or including **/scratch1** or **/scratch2**.
 2. Avoid having an executable located there. If possible, keep the executable in your home

space, in project space, or in other NFS-based scratch space.

Submitting Jobs: LSF on Q

Resource Management on ASCI Q Systems

- All LANL clusters use the Load Sharing Facility (LSF), a 3rd-party software from [Platform Computing Corporation](#) for resource management. But on the Q clusters, that's not all that is used.
- Hewlett Packard sells the AlphaServer systems used in LANL's Q clusters with another resource management system, called RMS (Resource Management System), which allows parallel programs to run on nodes across the Quadrics Interconnect.
- You need to know about both LSF and RMS in order to run big jobs on the LANL Q systems. However, there are only two important RMS commands: **prun** and **rinfo**



Advanced Info: [Typical Job Flow Scenario and How LSF and RMS Interact](#)

- You will use LSF utilities to gain interactive access to the back-end machines and to submit batch jobs for execution.
- The discussion in this class assumes that you are already familiar with LSF. If you are not, please see the [LSF tutorial](#).

A Closer Look at Q's Architecture

- Review: Each cluster (QA, QB, CA-B-C, CX, and QSC) consists of one or two *front ends* and some number of back-end *nodes*.
 - QA and QB: each contain 1,024 nodes; one front end for each
 - CA, CB, and CC: 128 nodes each; one front end for all
 - CX: 32 nodes; one front end
 - QSC: 256 nodes; two front ends
- Here's how the nodes are named:

Q Cluster Node Names			
Cluster	Nodes	Cluster	Nodes
QA	q0, q1, ..., q1023	CC	cc0, cc1, ..., cc127
QB	q1024, q1025, ..., q2047	CX	cx0, cx1, ..., cx31
CA	ca0, ca1, ..., ca127	QSC	qsc0, qsc1, ..., qsc255
CB	cb0, cb1, ..., cb127		

- Each cluster is actually composed of *domains*. A domain consists of 32 nodes.
- These domains share files and are the building blocks for the bigger clusters; i.e., each ASCI Q cluster consists of the tight coupling of multiple domains. Sometimes they're called "CFS domains" or "cluster domains."

- On all Q systems, LSF execution hosts are 32-node Tru64Cluster domains (see table below) with 4 nodes, 128 processors each. Submission hosts are either the 32-node domains or the front end(s). In other words, LSF basically knows nothing about individual compute nodes.

Q Cluster Domain Names		
Cluster	Domains / LSF Hosts	Front End(s)
QA	qd0, qd1, ... qd31	qfe1 (qafe)
QB	qd32, qd33, ... qd63	qfe2 (qbfe)
CA	cad0-cad3	cfe1 cfe2
CB	cbd0-cbd3	cfe1 cfe2
CD	ccd0-ccd3	cfe1 cfe2
CX	cx0	cxfe
QSC	qscd0, qscd1, ... qscd7	qscfe1 qscfe2

Submitting Jobs

- Use the LSF **bsub** command to submit batch jobs for execution. There are a multitude of options; see the LSF tutorial and the man pages.
- Use **llogin** to gain interactive access to the compute servers (back ends). Remember that **llogin** is essentially a script that submits a **bsub -Is ...** command and that it accepts many of the same options as **bsub**.
- Use Note: If you're a BlueMountain user make sure you eliminate any "**ptile**" options from your scripts - this will not work on the Q systems.

Running Parallel Codes

- **prun** is the RMS parallel job starter - it must be used if you want to run on more than one processor. This applies to MPI codes using the HP versions of MPI and to OpenMP codes. It does not apply to MPI codes using LAMPI.
- There are rarely any options used with **prun**. By default it will use the number of processors attached via LSF. Some available options are:

-n # Total number of processes for the job.
-N # Number of nodes.
-c # #cpus_per_process] (default is 1)
-t prefixes stdout with the process number
-vvv increased verbosity for possible error diagnosis

Examples:

bsub -n 4 prun a.out

Runs the code **a.out** on 4 processors of a single node.

bsub -n 16 prun -n 4 -c 4 mcnp5.mpi

This usage is for "hybrid" parallel computing models, using MPI and either threads or OpenMP. The example uses 4 MPI processes and 4 OpenMP threads per MPI process. The key is to request adequate resources from LSF for this, i.e., 4 * 4.

- The default allocation is one *processor* (or jobslot) on all LANL Q systems. Single-processor allocation is to allow more users access to resources but this means that jobs share memory and other system resources with other users on the node.

- If you want to run on multiple processors interactively then use

```
llogin -n # [other llogin options].
```

This will attach your interactive batch job to # processors. Then you can use **prun a.out** from the command line to run a parallel code. **prun a.out** will run the a.out executable with whatever number of processors you requested with **llogin**.

- You can attempt to log in to a particular segment of the CA/CB/CC cluster (after you're on **cfe1** or **cfe2**) using:

```
llogin -m segment_name
```

The **-m** option is either the segment name (ca, cb, or cc) or the corresponding domain name (cad0-3, cbd0-3, ccd0-3). You can also use a resource requirement string to achieve this (e.g., **-R ca**).

- (Important question to consider: Why might you prefer to use one C segment over another?
- (Alternate important question to consider: Why might you *not* preferentially choose one C segment over another when you **llogin**?

LSF Resources on ASCII Q

- Sample output from the **lshosts** command, which Shows static resource information for the machines, such as number of CPUs, total memory, total swap space, and LANL-defined LSF resources, is below.

Sample Output From lshosts Command on QA									
qafe-> lshosts									
HOST_NAME	type	model	cpuf	ncpus	maxmem	maxswp	server	RESOURCES	
qms0	ALPHA5	ES45	1.0	4	16074M	45954M	Yes	()	
qfe1	ALPHA5	DS20E	1.0	2	4003M	11297M	Yes	()	
qd0	ALPHA5	SCDOMAIN	1.0	128	8022M	10342M	Yes	(mem8	rms fs qa)
qd1	ALPHA5	SCDOMAIN	1.0	128	8022M	10342M	Yes	(mem8	rms cs qa)
qd2	ALPHA5	SCDOMAIN	2.0	128	8022M	10342M	Yes	(mem8	rms cs qa)
qd3	ALPHA5	SCDOMAIN	1.0	128	8022M	10342M	Yes	(mem8	rms cs qa)
qd4	ALPHA5	SCDOMAIN	1.0	128	8022M	10342M	Yes	(mem8	rms cs qa)
qd5	ALPHA5	SCDOMAIN	1.0	128	8022M	10342M	Yes	(mem8	rms cs qa)
qd6	ALPHA5	SCDOMAIN	1.0	128	8022M	10342M	Yes	(mem8	rms cs qa)
qd7	ALPHA5	SCDOMAIN	1.0	128	8022M	10342M	Yes	(mem8	rms cs qa)
qd8	ALPHA5	SCDOMAIN	1.0	128	8022M	10342M	Yes	(mem8	rms fs qa)
qd9	ALPHA5	SCDOMAIN	1.0	128	8022M	10342M	Yes	(mem8	rms cs qa)
qd10	ALPHA5	SCDOMAIN	1.0	128	8022M	10342M	Yes	(mem8	rms cs qa)
qd11	ALPHA5	SCDOMAIN	1.0	128	8022M	10342M	Yes	(mem8	rms cs qa)
qd12	ALPHA5	SCDOMAIN	1.0	128	8022M	10342M	Yes	(mem8	rms cs qa)
qd13	ALPHA5	SCDOMAIN	1.0	128	8022M	10342M	Yes	(mem8	rms cs qa)
qd14	ALPHA5	SCDOMAIN	1.0	128	8022M	10342M	Yes	(mem8	rms cs qa)
qd15	ALPHA5	SCDOMAIN	1.0	128	8022M	10342M	Yes	(mem8	rms cs qa)
qd16	ALPHA5	SCDOMAIN	1.0	128	8022M	10342M	Yes	(mem8	rms fs qa)
qd17	ALPHA5	SCDOMAIN	1.0	128	8022M	10342M	Yes	(mem8	rms cs qa)
qd18	ALPHA5	SCDOMAIN	1.0	128	8022M	10342M	Yes	(mem8	rms cs qa)
qd19	ALPHA5	SCDOMAIN	1.0	128	8022M	10342M	Yes	(mem8	rms cs qa)
qd20	ALPHA5	SCDOMAIN	1.0	128	8022M	10342M	Yes	(mem8	rms cs qa)
qd21	ALPHA5	SCDOMAIN	1.0	128	8022M	10342M	Yes	(mem8	rms cs qa)
qd22	ALPHA5	SCDOMAIN	1.0	128	32768M	45075M	Yes	(mem16	mem32 rms cs qa)
qd23	ALPHA5	SCDOMAIN	1.0	128	32768M	45075M	Yes	(mem16	mem32 rms cs qa)
qd24	ALPHA5	SCDOMAIN	1.0	128	8022M	10342M	Yes	(mem8	rms fs qa)
qd25	ALPHA5	SCDOMAIN	1.0	128	8022M	10342M	Yes	(mem8	rms cs qa)
qd26	ALPHA5	SCDOMAIN	1.0	128	16384M	45075M	Yes	(mem16	rms cs qa)
qd27	ALPHA5	SCDOMAIN	1.0	128	16384M	45075M	Yes	(mem16	rms cs qa)
qd28	ALPHA5	SCDOMAIN	1.0	128	16384M	45075M	Yes	(mem16	rms cs qa)
qd29	ALPHA5	SCDOMAIN	1.0	128	16384M	45075M	Yes	(mem16	rms cs qa)
qd30	ALPHA5	SCDOMAIN	1.0	128	16384M	45075M	Yes	(mem16	rms cs qa)
qd31	ALPHA5	SCDOMAIN	1.0	128	16384M	45075M	Yes	(mem16	rms cs qa)
qafe-> lshosts -R mem32									
qd22	ALPHA5	SCDOMAIN	1.0	128	32768M	45075M	Yes	(mem16	mem32 rms cs qa)
qd23	ALPHA5	SCDOMAIN	1.0	128	32768M	45075M	Yes	(mem16	mem32 rms cs qa)



Sample output from lshosts command on the CX front end

```

cxfe-> lshosts
HOST_NAME      type      model  cpuf  ncpus  maxmem  maxswp  server  RESOURCES
cxms           ALPHA5    DS20E   1.0    2    989M   4287M   Yes  ()
cxfe           ALPHA5    DS20E   1.0    2   4003M  11297M   Yes  ()
cxd0           ALPHA5    SCDOMAIN 1.0   128   4096M  10342M   Yes  (rms)

```



Sample output from lshosts on the CA, CB, or CC front end

```

cfe1-> lshosts
HOST_NAME      type      model  cpuf  ncpus  maxmem  maxswp  server  RESOURCES
cserver        ALPHA5    DS20E   1.0    1    989M   3146M   Yes  ()
cams           ALPHA5    DS20E   1.0    2    989M  11508M   Yes  ()
cfe1           ALPHA5    DS20E   1.0    2   4003M  11516M   Yes  ()
cfe2           ALPHA5    DS20E   1.0    2    980M   4113M   Yes  ()
cbms           ALPHA5    DS20E   1.0    2    989M  11508M   Yes  ()
ccms           ALPHA5    DS20E   1.0    2    989M  45956M   Yes  ()
cad0           ALPHA5    SCDOMAIN 1.0   120   3994M  10342M   Yes  (rms ca)
cad1           ALPHA5    SCDOMAIN 1.0   120   4096M  10342M   Yes  (rms ca)
cad2           ALPHA5    SCDOMAIN 1.0   120   3998M  10342M   Yes  (rms ca)
cad3           ALPHA5    SCDOMAIN 1.0   120   3994M  10342M   Yes  (rms ca)
cbd0           ALPHA5    SCDOMAIN 1.0   120   4096M  10342M   Yes  (rms cb)
cbd1           ALPHA5    SCDOMAIN 1.0   120   3994M  10342M   Yes  (rms cb)
cbd2           ALPHA5    SCDOMAIN 1.0   120   4096M  10342M   Yes  (rms cb)
cbd3           ALPHA5    SCDOMAIN 1.0   120   3994M  10342M   Yes  (rms cb)
ccd0           ALPHA5    SCDOMAIN 1.0   120   3994M  10342M   Yes  (rms cc)
ccd1           ALPHA5    SCDOMAIN 1.0   120   3994M  10342M   Yes  (rms cc)
ccd2           ALPHA5    SCDOMAIN 1.0   120   3994M  10342M   Yes  (rms cc)
ccd3           ALPHA5    SCDOMAIN 1.0   120   3994M  10342M   Yes  (rms cc)

```

- **bmgroup** The 32-node domains are organized into LSF host groups and LSF queues dispatch jobs to the LSF host groups. This command shows how the LSF host groups map to the Q domains in the cluster. The definitions are generally recursive - at the higher levels the names are all of the form "xxx-m" and at the lower levels are the domain names - and there are also some duplicate definitions.
- Note that queues don't necessarily have exclusive access to the LSF host groups.




Sample Output From bmgroup Command on QA

```

qafe% > bmgroup
GROUP_NAME      HOSTS
file-m          qd0 qd24 qd16 qd8
filellnl-m      qd0 qd8
filesnl-m       qd24 qd16
compute-m       qd30 qd10 qd20 qd31 qd1 qd11 qd21 qd2 qd12 qd22 qd3 qd13 qd23
qd4 qd14 qd5    qd15 qd25 qd6 qd26 qd7 qd17 qd27 qd18 qd28 qd9 qd19 qd29
qa              file-m/ compute-m/
snl.llnl-m      file-m/ compute-m/
llnl-m          snl.llnl-m/
snl-m           snl.llnl-m/
standby-m       snl.llnl-m/
drm-m           compute-m/
full-m          compute-m/ file-m/
test-m          full-m/
dat-m           full-m/

```

Sample output from bmggroup command on CA

```

cfe1> bmggroup
GROUP_NAME  HOSTS
ca          cad0 cad1 cad2 cad3
cb          cbd0 cbd1 cbd2 cbd3
cc          ccd0 ccd1 ccd2 ccd3
testa-m     ca/
data-m      ca/
ca-m        ca/
cb-m        cb/
cc-m        cc/
testb-m     cb/
testc-m     cc/
datb-m      cb/
datc-m      cc/

```

- **LSF Resource Requirement Strings** contain lists of LSF resources or logical expressions involving LSF resources. They are used via the `-R resource_requirement_string` option to restrict LSF commands so that they run on or produce output limited to only those identified resources.
- Most commonly this is used on the Q systems to submit batch jobs to domains containing nodes with more memory. They can also be used to select specific domains (which might be useful to optimize I/O performance; see the I/O section for more details).

Examples:

<code>bsub -R "hname==qd39" a.out</code>	Request domain qd39 on QA
<code>bsub -R "mem16" ...</code>	Request one of the eight domains on QA and QB containing 16-GB nodes.
<code>bsub -R "mem32" ...</code>	Request one of the four domains on QA and QB containing 32-GB nodes. Note that these domains also serve as 16-GB LSF resources (we think).

Monitoring Jobs With RMS

- In addition to the LSF job monitor command [bjobs](#), you can also monitor parallel jobs and the status of nodes and resources within RMS. Do this with the [rinfo](#) command.
 - `rinfo` only works on the back ends.
 - An example of `rinfo` output is [here](#). This output was produced with no command options and it gives full output. You can obtain a highly abbreviated summary of machine parameters using `rinfo -nl`.

Four parts to the `rinfo` output: (1) Machine Name; (2) Partition names (LANL systems always have 2 partitions, "root" and "full"); (3) resources allocated to jobs; and (4) jobs running.

Modifying Jobs

- **`bkill`** Sends a Unix signal to a running job. (see `/usr/include/signal.h`)

Examples:

```

bkill 754261
bkill -s 9 754261
bkill -s SIGURG 754261

```

- On the Q systems the `bkill` command has a `-r` option. Use `bkill -r` only on jobs that cannot be killed in the operating system, or on jobs that cannot be otherwise removed using `bkill`.

Error Messages

- If your LSF job dies abnormally and you want to find out why there are several steps to follow. The first is to use the **bhist** command. However, interpreting the result depends on whether the job died because of LSF, or **prun**, or something shell-related. It also depends on which shell you were using. There are detailed instructions for this on <http://computing.lanl.gov/article/178.html>.

Creating and Using Job Scripts

- You can use **bsub** either with a command line interface or you can enclose its options in a job control script. A job control script is nothing more than a shell script file that includes LSF instructions. This file is submitted to LSF via the **bsub < script_file** command. That is, you redirect the script file as input to **bsub** .
- The script should reside in the directory from which you submit the **bsub** command.
- See the [LSF tutorial for more information on using scripts.](#)
- You must remember to redirect the job script as stdin! If you don't use the redirect, the **#BSUB** options are not parsed by bsub. Also, redirecting the script spools the entire script at the time of the job submission, so that any change to the script between the submit time and the run time is not picked up by the job.
- For more detailed training in scripts, we recommend taking an Advanced Unix course In the meantime, the LANL ICN Consultants (5-4444), are happy to help out with script questions.

Q's Queue Structure

- NOTE: Any queue info given in this document is for illustrative purposes only. Queue parameters can and do change frequently. In order to see the most up-to-date information run the [bqueues](#) command on the cluster of interest.
- To see what the default queues are on a Q system use the LSF command **bparams**.
- Most of the LANL Q clusters have these queues:
 - **intq** or **devq** - development on File Servers, few procs, 12 hrs. On QA, QB, CX, and QSC it's DEVQ; on CA/CB/CC it's INTQ.
 - **smallq** or **largeq** - small or large refers to # of processors; production jobs on compute servers
- A new queue, big1q, was created on QB on Friday, Feb 20, 2004. This queue has 15 dedicated domains, taken from the 27 that formerly were available to dswlq, largeq, smallq, and dswq. Thus, resources for these queues are now significantly limited.
- On QSC each user is limited to 4 processors in devq. Devq is restricted to Domain 0 (qscd0). Other domains, including qscd7, run batch jobs.
- As of March, 2005, one-third of QA is available to LANL users via two queues, largeq and devq, the latter for interactive work. As of 28 July LANL share is 47.5%.

Queue Characteristics For Machine QA							
Queue	Priority	CPUs per Job Def/Max	RUNLIMIT Def/Max (hours)	Queue Job Limit	User Job Limit	FairShare	Notes
sdevq	7	4/8	12/12	128	8	no	default queue
svizq	7	4/128	12/12	128	128	no	
ldevq	7	4/8	12/12	128	8	no	default queue
lvizq	7	4/128	12/12	128	128	no	
snlq	4	4/1792	4/96	1792	1792	yes	default queue
lsmallq	4	1/1	12/12	128	1	yes	default queue;

							BACKFILL
llongq	4	1/1	12/12	128	1	yes	BACKFILL
llargeq	4	1/1	12/12	128	1	yes	default queue; BACKFILL
lhighq	4	1/1	12/12	128	1	no	
standbyqa	1	4/3584	-/-	128	3584	no	PREEMPTABLE

- Note: This info is provided for illustrative purposes . Queue parameters can change from time to time.

Queue Characteristics For Machine CX							
Queue	Priority	CPUs per Job Default/Max	RUNLIMIT Def/Max (hours)	Queue Job Limit	User Job Limit	FairShare	Notes
devq	6	1/1	12/12	128	1	no	Interactive Only
longq	5	1/4	6/48	16	8	no	Interactive & Batch
smallq	5	1/4	4/12	80	-	no	Interactive & Batch

Shared Queue Characteristics For Machines CA, CB, CC							
Queue	Priority	CPUs per Job Def/Max	RUNLIMIT Def/Max (hours)	Queue Job Limit	User Job Limit	FairShare	Notes
intq	6	4/4	6/6	84	4	no	Interactive & Batch
highq	5	4/52	4/45	52	-	Fairshare among milestone users	Interactive & Batch
smallq	4	4/256	4/16	1344	-	same as QB smallq	Interactive & Batch

Exercise

Practice Using LSF

- If you're on a front end then submit an **llogin** command. Answer the following questions:
 - Which LSF host are you on?
 - Which node are you on?
 - Which queue did your job get submitted to?
- Use the **bhosts** command to determine LSF host status. Does everything look normal? Remember that **bhosts -w** gives more information. How busy is the machine? What's another LSF command you can use to determine system load?
- Type **bqueues** and then **bqueues -u user** where **user** is your user ID. Which queues are you allowed to use?
- Use **bjobs** to find out what LSF jobs you have running now. Make sure you understand the output.
- To get a different view of what's going on try the [rinfo](#) command.
- The following instructions involve running a code. It is an MPI-parallel code that you want to run on 2 processors. You can run it interactively or in batch mode; your choice, but doing both would

be very instructive. If you try the batch mode, writing a short script would be best. Here are basic instructions but you need to figure out exactly what to do.

- The instructor has given you a tar file using the **give** utility on one of the Q Tru64 systems. Retrieve the file and copy it to a file space in which you want to work (your choice of where). The file is Q-class.tar.
- Untar it (**tar xvf *.tar**). The executable has been prepared ("sweep-mpi.exe"). There's also an input file required ("input").
- Run this code on your own on 2 processors. It produces output on the terminal. You'll know it ran if there's a CPU and Elapsed time printed. The next paragraphs contain hints. Try to run the code before looking at them. Think about what you need to do.

When someone gives you a file using **give** the result is in **/givedir/your_user_id**. Copy the code from there.

Because this is an MPI code you need to load an MPI modulefile before you can run it. The error message **/sbin/loader: Fatal Error: Cannot map library libfmpi.so** is due to this. The code was made using the modulefile **MPI_default**. That's the one you need.

Another error message you might see is "tasks do not match mpi_comm_size." This comes from the code and may be due to two conditions. First, remember that you need to use "**prun**" to launch a parallel job, as in "**prun sweep-mpi.exe**." If you don't use "**prun**" the system runs your job on one processor. Sweep wants 2 and so you get this error.

But even if you use "**prun**" you might get this error, probably because Sweep wants 2 processors but maybe you don't have 2 allocated to you, particularly if you're trying to run interactively. If you are trying to run interactively figure out how many processors you have allocated to you using **bjobs**. If it's less than 2 then log out (to the front end) and **llogin** again, this time requesting 2 processors. When you get in, run the code.

Try running the code in batch mode by creating a script containing **something like**

```
bsub -n 2 -e ouch -o outp -W 0:05 -J MySweep ./sweep-mpi.exe.
```

Make sure you understand what this line is doing and what it might be missing. Make sure you remember how to submit a BSUB script. Check the output to make sure it ran. Follow its progress with **bjobs** and/or **bjobs -lp** and/or **bpeek** (although you have to be quick to use the **bpeek** because the code runs quickly). Try it several times, if need be.

If you have any problems or questions don't hesitate to contact the instructor.

- There's also a batch submission script in the tar file, called "job" that you can use.
- This concludes the LSF exercises.



Contents

Compiling

- You need to load the compiler's modulefile for FORTRAN, C, and C++
- Note: The OS upgrade (to Eagle) is taking place during Q1 2004. It is strongly recommended that you recompile all codes due to new libraries.

Porting code from IRIX to Tru64

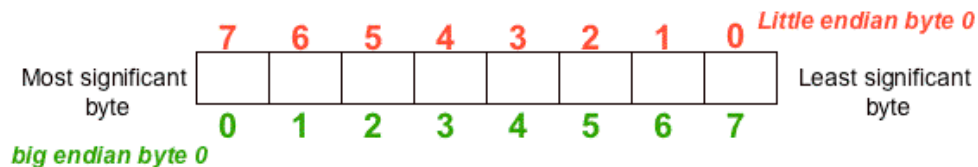
Big Endian vs. Little Endian

- This refers to the order in which the machine stores the bits of a word.

Byte Order of Some Important Systems		
Machine	Processor	Byte order
ASC Q	DEC/Compaq/HP Alpha EV68	Little Endian
Lambda, Pink, Grendels	Intel Pentium	Little Endian
Blue Mountain	MIPS R10000	Big Endian
Mauve	Intel Itanium	Little Endian
Lightning, Flash	AMD Opteron	Little Endian
ASC White	IBM Power	Big Endian

Addressing Objects: Endianess

- **Big Endian:** address of most significant byte = word address (big end of the word).
 - 0xDEADBEEF = DE AD BE EF
 - IBM 360, Power; Motorola 68k; MIPS; SPARC; HP PA
- **Little Endian:** address of least significant byte = word address (little end of the word)
 - 0xDEADBEEF = EF BE AD DE
 - Intel 80x86; DEC Vax; DEC Alpha



- Two basic Endian-related issues: Code or compilation issues and runtime issues.
 - Code or compilation issues
 - Dereferencing pointers on Blue Mountain vs. Q. Thanks to John Daly for providing this example. Consider the following C program:

```
#include main()
{
    long *a;
    int *b;
    a = (long *) malloc(8);
    a[0] = 0x1234567890abcdef;
    b = (int *)a;
    printf("A: %lx, B: %lx %lx\n", a[0], b[0], b[1]);
}
```

On Blue Mountain this program yields:



Pointer Dereferencing on Blue Mountain

```
t01-> cc -64 -o end.theta.64 end.c
cc-1178 cc: WARNING File = end.c, Line = 9
      Argument is incompatible with the corresponding format
      string conversion.

      printf("A: %lx, B: %lx %lx\n", a[0], b[0], b[1]);
                                     ^

cc-1178 cc: WARNING File = end.c, Line = 9
      Argument is incompatible with the corresponding format
      string conversion.

      printf("A: %lx, B: %lx %lx\n", a[0], b[0], b[1]);
                                     ^
t01-> end.theta.64
A: 1234567890abcdef, B: 12345678 ffffffff90abcde
```

On Q this program yields:



Pointer Dereferencing on Q

```
qsc10-> cc end.c -o end.q
qsc10-> end.q
A: 1234567890abcdef, B: ffffffff90abcdef 12345678
```

■ Runtime issues

- Data files written out on Blue Mountain must be converted to little-Endian format before they can be read on Q systems.
- Let's say you have a data file that you created on machine Theta or Blue Mountain and you want to read that data file in to a FORTRAN program on one of the Q systems. To convert the data file from big- to little-Endian in FORTRAN you can:
 - Add "**convert=BIG_ENDIAN**" to the file OPEN statement.
 - Or, compile with **-convert big_endian**.
 - Or, if you don't have access to the code or if you want to do the conversion on a file-by-file basis, you can set an environment variable: **setenv FORT_CONVERTn big_endian** where *n* is the FORTRAN I/O unit number for the file you're opening.
- To Convert a data file from big- to little-Endian in C you have to write your own routine to reverse the bit order. Or you can use the one the consultants have [here](#).

FORTRAN

- Compiler front ends are **f77**, **f90**, and **f95**. The [man page](#) is the same for all. This is HP's compiler.
- The compiler is extremely fast and is generally more adherent to the standard than the MIPS FORTRAN compiler was.
- The optimized math libraries (BLAS, LAPACK, FFTs, other solvers, etc.) are in **-lcnml** for both C and FORTRAN. Complete [documentation](#) is available in the open partition.
- Compiler optimization levels are:
 - **-O0**: no optimization.

- : Local optimization: straight-line code.
 - **-O2**: Optimizations that do not significantly increase code size: code motion; strength reduction, induction variables, and test replacement; split lifetime analysis; scheduling.
 - **-O3**: Everything from -O2 plus loop unrolling, final code replication.
 - **-O4**: Everything from -O3 plus function inlining plus software pipelining.
 - **-O5**: Everything from -O4 speculate, loop transformations and byte vectorization.
 - Note that **-O4** is the default, unless you specify certain debugging options (**-g2 -g -gen_feedback**).
- For best optimization results use: **-fast**. This turns on:
 - **-O4**
 - **-arch host**: use instructions of host machine
 - **-tune host**: rearrange instructions to run best on host machine
 - **-math_library fast**: use alternate, faster math library routines for intrinsic functions, such as SQRT, EXP, LOG, LOG10, SIN, and COS
 - **-assume noaccuracy_sensitive**: allow reordering of floating point expressions
 - **-align dcommons**: align entities in COMMON on natural boundaries up to 8 bytes
 - Here's an example of how performance relates to optimization level, in case you need to back off from the highest opt level.

Runtimes vs. Opt Level	
FORTRAN Opt Level	Time (seconds)
-O0	10.3
-O1	5.2
-O2	2.5
-O3	2.2
-fast	1.8

- The compiler reference manuals are available at the public internet URL: <http://www.compaq.com/FORTRAN/docs> under the heading "HP FORTRAN for Tru64 UNIX."
- Here are some other useful options:
 - **-fixed** : Interprets FORTRAN source files as fixed (FORTRAN 77 style) source format.
 - **-fpe0 or -fpe** : Terminates a program under for certain types of floating-point operation errors. Interestingly, this is the DEFAULT behavior on the Q systems. (It wasn't the IRIX MIPSpro compiler default behavior.)
 - **-g3**: Allows debugging but does not change the optimization level.
 - **-check bounds -r8**: Defines REAL declarations, constants, functions, and intrinsics as DOUBLE PRECISION (REAL*8)
 - **-V**: Creates a source listing file with various compile-time information appended. The name of the listing file is the basename of the source file with a .l suffix. The default is -noV

C and C++

- For Compaq/HP C load the module **HP_CC_6.5-027** or something that looks like it. The compiler is 'cc.'
- For C++ there are three choices: CXX (from Compaq/HP), GCC (Gnu), and KCC from (Kuck & Assoc.), each available in their own modulefile.
- For more information on code development see the "compile code" section for each of the machines on computing.lanl.gov.

Compiling MPI Codes

- An MPI modulefile must be loaded.
- On the Q systems there are generally several MPI packages available, and for some of the packages, several versions available. The packages are HP's version of MPICH (an open-source MPI from Argonne NL), "Alaska" (the thread-safe, proprietary, tuned version from HP), and LAMPI, a LANL-authored package.

- Currently you will find something like:
 - MPI_64bit_R6: MPICH
 - MPI_64bit_R13: Thread-safe Alaska
 - MPI_default: points to MPI_64bit_R13
- You need an MPI header file in your code:
 - FORTRAN: `INCLUDE 'mpif.h'`
 - C: `#include "mpi.h";`
- As is the case with other LANL systems your compile lines need to include the path to the MPI include files and load libraries.
- If your MAIN is a FORTRAN code, you need to load with `-lfmpi`.
If your MAIN is a C code, you need to load with `-lmpi`.
- Here's an example using FORTRAN:

```
[hjl@qsc3 ~/SWEEP]$ module list
No Modulefiles Currently Loaded.

[hjl@qsc3]$ module load fortran_default
[hjl@qsc3]$ module load MPI_64bit_Thread_Safe_R13

[hjl@qsc3]$ f90 -lfmpi $MPI_COMPILE_FLAGS $MPI_LD_FLAGS my_mpi_code.f90
```

- If you use MPI-IO, you need to load with `-lmpio` (both FORTRAN and C).
- While we're on the subject of MPI, there are four environment variables that you might want to know about:
 - **LIBELAN_TPORT_BIGMSG=bytes**. Messages that are larger (in bytes) than the value of LIBELAN_TPORT_BIGMSG are sent only when a matching receive has been posted. The transfer is synchronous and the receiver can limit the size of receive message buffers. The default value of the variable is 4 Mbytes. Messages smaller than the given value use eager protocol.
 - **LIBELAN_SHM_ENABLE=1** - Enables or disables communications within a node being transferred via shared memory. The default value is TRUE (1)
 - **LIBELAN_ALLOC_SIZE=bytes** - Defines the amount of virtual memory (in bytes) that is allocated for use by the MPI system buffer pool. The default value is 200MBytes.
 - **LIBELAN_WAITTYPE WAIT|POLL|n**. Determines the default behavior as either interrupt mode or polling mode MPI. The value of **n** switches from polling to interrupt after **n** tries.
 - For more info on using these variables and on MPI message passing performance see the [Compaq \(HP\) presentation](#).

Exercise

Practice Compiling FORTRAN

- The Sweep tar file should contain the following source files:


```
sweep-single.f
sweep-mpi.f
timers.c
```
- Compile the code to run as a sequential binary. Use something like:


```
f90 -fast -w sweep-single timers.c -o sweep-single
```

- Compile the code to run as an MPI binary. Use `sweep-mpi.f` as the source (and `timers.c`). You have to modify the compile line. Contact the instructor if you need help.
- This concludes the compiling exercises

Contents

Using OpenMP on Q

- [OpenMP](#) is an extension to standard FORTRAN, C and C++ to support a global shared address space parallel computing model. Directives have to be added to your source code to parallelize loops and specify certain properties of variables.

Compiling and linking

- FORTRAN with OpenMP directives is compiled as:

```
% f90 -fast -omp myprog.f -o myprog.exe
```
- C code with OpenMP directives is compiled as:

```
% cc -fast -omp myprog.c -o myprog.exe
```

Running OpenMP jobs

- To run an OpenMP job interactively, there are 3 steps:
 1. `llogin -n 4`
 2. Set the `OMP_NUM_THREADS` environment variable:

```
% setenv OMP_NUM_THREADS 4
```
 3. Then run the executable:

```
% ./myprog.exe
```
- The last two steps can be combined:

```
% env OMP_NUM_THREADS=4 a.out
```
- For larger jobs and production use, submit a job to the LSF with something like

```
% bsub < omp_script
```

 where `omp_script` contains something like

```
#!/bin/tcsh
setenv OMP_NUM_THREADS 4
./a.out
```
- For more info on Tru64 OpenMP see
 - <http://h18009.www1.hp.com/fortran/docs/lrm/dflrm.htm> is the FORTRAN Language Reference Manual and Appendix F.2.1 lists the OpenMP FORTRAN API Run-Time Library Routines. Section 15.2.3 is all about OpenMP.
 - <http://h18009.www1.hp.com/fortran/docs/unix-um/dfumtoc.htm> is the Compaq FORTRAN User Manual; Chapter 6 is all about OpenMP and it is quite extensive. Appendix D in this manual is also relevant.
 - Also recommended is www.openmp.org on which you can find the spec and

Debugging on Q

- First, don't forget where core files are produced: in `/local/core/rms/rms_id`, where *rms_id* is the RMS resource you were allocated. Use `rinfo` and look for your userid to find it.
- **TotalView** - GUI-based debugger that can be used for serial and all types of parallel code. This is the official ASCI debugging system. TotalView is integrated with RMS and the HP MPI libraries. Using TotalView is covered in detail in the [TotalView tutorial](#), which is taught here at LANL on an ongoing basis. Go to <http://asci-training> to register.
- To use TotalView on Q:
 - Remember to compile with `-g` or something similar.
 - You need to load a TotalView modulefile.
 - To run TotalView on Q do the following. (Note: these examples assume you've used LSF first.)
 - simple serial code:

```
totalview a.out
```
 - simple serial code with a core file:

```
totalview a.out core
```
 - simple serial code with options to your code:

```
totalview a.out -a <options to a.out>
```
 - MPI code on Q machines:

```
totalview prun -a <prun options> a.out
```

Type 'g' or click the 'Go' button the process window (If your code is large or you are running with many processes, you may see your code stopping at various breakpoints. It will continue without intervention.)

To set breakpoints in your code, answer 'yes' to the question "Process prun is a parallel job. Do you want to stop the job now?"
 - Interesting: If an MPI code dumps core and you just want to find out where it was executing you can still do `totalview a.out core`. You will not be able to execute.
- **Ladebug** - This is HP's standard debugger that comes with Tru64 Unix. It does source-level debugging using both a command line and GUI interface for serial, multi-threaded, and multiprocess applications.

Complete documentation for ladebug is available from HP [here](#). See the man page [here](#).

- **dbx** - This is the standard Unix command line debugger with some thread support. See the man page [here](#). For very quick, simple debugging tasks, dbx may be recommended.

- LANL's [ICN Consulting Office](#) will handle questions on Q. You can reach them at 5-4444, option "3", or send email to consult@lanl.gov
- Don't forget about these three important web sites:
 1. LANL's main computing documentation page: <http://computing.lanl.gov>
 - Did you know that you can search computing.lanl.gov?

In the open, simply placing a term after <http://computing.lanl.gov/> will perform a basic search (as long as it doesn't actually match the name of a legitimate page (i.e. <http://computing.lanl.gov/lst>)

Note that this will not work in the secure.
 - Also, the Lab's google search does a great job indexing the computing.lanl.gov site. To restrict results to this site while using the google search on the Lab's homepage, add "site:computing.lanl.gov" to your search term. The [Advanced Search](#) method is recommended.
 2. LANL's main computing status page: <http://icnn.lanl.gov>
 3. Main HPC Training page: <http://asci-training.lanl.gov>
- Another monitoring tool you should know about is **DRM**, the [Distributed Resource Monitoring tool](#), available in both the secure and yellow partitions on <http://icnn.lanl.gov>. Try it! In the secure a better web address for it is <http://icnn/drm/alljobs>.

[Contents](#)

Reference Info

- [The main site for HP technical computing.](http://www.hp.com/techservers/software/index.html) <http://www.hp.com/techservers/software/index.html>
- [Online documentation for HP's FORTRAN compiler.](http://h18009.www1.hp.com/fortran/docs/index.html) <http://h18009.www1.hp.com/fortran/docs/index.html>
- [on-line TotalView information](#)

[Contents](#)

The next part of this course is [here](#).

LA-UR 04-3787 (June, 2004)



Operated by the [University of California](#) for the [National Nuclear Security Administration](#), of the US [Department of Energy](#). [Copyright © 2004 UC](#) | [Disclaimer/Privacy](#)

Contact Us : consult@lanl.gov

Last Modified: March, 2005



TABLE OF CONTENTS

More LSF Topics

- Reservation & Backfill
- Fairshare

Architecture

- General
- Alpha Processor
- Quadrics
- Storage

I/O on Q

- File management
- Small Operations
- Large Serial Files
- Parallel N to N
- Parallel N to 1
- Parallel Copies
- HPSS Performance

Performance Analysis

- Timing a Code
- gprof
- Pixie
- HiProf
- uprofile
- Misc. Tools

Where to Go for Help

Course Evaluation

Introduction to ASCI Q - Advanced

Additional LSF Topics

Processor Reservation and Backfilling

- Consider a simple case - several jobs in a queue waiting to run on a 128-processor machine.

Queue				Machine
job	# CPUs	Runlimit	Status	
#1	100	45 min	Running	128 processors: 100 in use by job #1 28 available Job #1 will finish in 45 minutes
#2	64	60 min	Pending	
#3	16	30 min	Pending	
#4	32	30 min	Pending	

- Without processor reservation:**

Job #2 can't run, job #3 schedules ahead of it.
Result: Large jobs tend to starve.

- With processor reservation:**

Job #2 reserves all available processors and grabs others when they become available.
Result: idle processors (while job #1 finishes).

- With processor reservation & backfilling:**

Job #2 reserves available processors but job #3 is allowed to use 16 of the reserved processors since it can complete before job #2 will be able to run anyway.

- Moral: Try to accurately estimate your resource requirements.**

LSF and Fairshare

- In the absence of fairshare LSF dispatches jobs to run based on the priority of the queue and a first-come, first-served order within the queue.
- This has obvious limitations. For example, one user could submit many long jobs at once and monopolize the cluster's resources for a long time.
- To prevent this, we can use fairshare scheduling to control how resources should be shared by competing users. Fairshare determines which jobs are selected to run based on three parameters:
 1. Predetermined (static) allocations for users or groups of users;
 2. committed run time, including job slots reserved and in use;
 3. and a decaying history of recent usage.



- There are two ways fairshare can be implemented at LANL
 - **Queue-based:** Priorities are calculated based on usage in a given queue, even if the queue shares hosts with other queues. Examples: BlueMountain, QB, CA/CB/CC, QSC.
 - Sometimes called "cross-queue" fairshare.
 - To get fairshare information on a queue-based system use the LSF command
`bqueues -l`
 or `bqueues -lr` if you really want a lot of output. There's no single command to tell you which queues have fairshare; you have to use
`bqueues -l | egrep "QUEUE|FAIR".`
 - **Host-based:** Priorities are calculated based on usage of all hosts in the cluster even if though workload for these hosts will come from multiple queues. Examples: QA, CX*, Theta, Lambda
 - To get fairshare information on a host-based system use the LSF command
`bhpart`
 or `bhpart -r` if you really want a lot of output. If you run `bhpart` on a queue-based fairshare system it will return the message "No host partition defined in the system."
 - It's possible to set up fairshare with equal shares for all users (no single user can dominate); done this way on LANL machine CX.
- The output from the above two commands basically has three parts, two of which are shown in the example below. The third part, not shown, is a long list of shares for each user on the system.

FAIRSHARE_FACTORS -- time units: hours RUN_JOB_FACTOR: 0.00 CPU_TIME_FACTOR: 0.00 RUN_TIME_FACTOR: 1.00 COMMITTED_RUN_TIME_FACTOR: 1.00 Historical Run Time is enabled in Fairshare.								
SHARE_INFO_FOR: QSC_CLUSTER/								
USER/GROUP	SHARES	PRIORITY	STRTD	RSV	CPU_TIME	RUN_TIME	HIST_RTIME	COMM_RTIME
adwp	25000	58.53	0	0	49553.7	0.0	427.1	0.0
allnc-asci	20000	43.80	16	0	7307.8	13.1	392.7	64.0
institution	35000	12.45	512	0	21220.4	278.8	762.6	2048.0
others	20000	11.86	128	112	47483.5	116.4	1174.5	512.0

- The first part of the output shows the fairshare factors used in the formula to assign job priorities. The formula may be viewed [here](#). The factors are set in the LSF configuration files and are not changed often so this part of the output remains static. All clusters at LANL are configured with the factors as shown in this example.
- The second part of the output contains mostly dynamic information, including the calculated priorities, which are essentially instantaneous and constantly changing because of changing job loads. The components of this part of the output are as follows:

USER/GROUP

These are the groups for which static shares are allocated (adwp, allnc-ascii, institution, and others, in the example above).

To find out "who" is in a group use the LSF command **bugroup**, which recursively lists all groups, or **bugroup <group_name>** where **<group_name>**, which just lists the members of one group.

To find which share groups you're in (or another user is in) use the LSF **lsfgroups [user_id]** command.

SHARES

The (statically-allocated) user/group shares.

PRIORITY

The (instantaneous) LSF job priority.

STRTD/RSV

The number of job slots currently running or reserved by users of the host partition or queue.

CPU_TIME

The cumulative CPU usage across the host partition or queue, modified by a decay factor. The decay factor is set so that basically only relatively recent CPU activity counts. At LANL it is set so that after the system's historical time, usage counts as 10%. To see the historical time set for a system, run the command "**bparams -1 | grep HIST_HOURS**".

RUN_TIME

This is the wallclock time accumulated by jobs currently running, calculated as the wallclock time of a job times the number of processors used by the job (time x numprocs).

HIST_RUNTIME

This is the wallclock time of finished jobs from the partition or queue (in hours), modified by the same decay factor as the CPU time. Like RUN_TIME, it is also calculated as (time x numprocs).

COMM_RUNTIME

This is the remaining committed wall-clock time of current jobs (in hours). This value decreases as run times increase. Including committed time prevents burst scheduling - running numerous jobs from a single high-priority user. Each running job affects the user's (and or group's)priority as though the job has run its full committed time.

IMPORTANT: The fairshare formula charges your Fairshare group for the requested run time when your job starts running. Specifying longer run times reduces the shares available for you and other users in your group. This relates to the "moral" given above: If you don't specify the -W option the queue's RUN_LIMIT is used as the committed runtime for fairshare purposes.

Which Job Will Run Next?

- LSF makes the machine info available to you but it's still very difficult (if not impossible) to determine which job will run next at any given time, because the systems - the queues of jobs waiting to run and the load currently running - are so dynamic. Any two successive looks may tell a different story. The priority you see is only an approximation.

For example, a job which terminates early or one that backfills on reserved processors affects the priority of its share group, thus affecting the relative priorities of all

pending jobs.

- Also, you may need to look beyond a single queue. With queue-based fairshare, you need to consider the relative queue priorities in addition to job priorities. With host-based fairshare, the highest-priority job from each queue running on the host partition must be compared.
- A group of users may use more than their static share of a host-based fairshare cluster if the other groups aren't using their allocation.
- One thing that can be said with certainty: when a job starts reserving processors it is likely to run soon. Use the LSF **reserved** or **reserved -f** commands to see which LSF jobs are currently reserving processors.



Example LSF reserved Command Output

```
qscfel% reserved -f
```

```
Jobs reserving processors at Wed Apr 20 12:16:17 MDT 2005 are:
```

jobID	User	#Reserved	#Requested	StartTime	Queue	Domains	Reserving
72407	zuhone	60	256	Apr 20 16:31	largeq	<60*qscd7>	

- Fairshare may be imprecise in telling you when your job will run but it can help explain why other jobs run ahead of yours. A common complaint, for example, involves a user who submits a job that pends for several days while many other jobs of similar size run. The user with the pending job probably has a greater HIST_RUNTIME.
- A similar complaint involves users who belong to a group with a large static share whose jobs pend while jobs from lower-share users run.
- Also, the number of jobs a user has pending at any given time has no bearing on the likelihood of any one of those jobs being dispatched to run. The fairshare formula that calculates instantaneous priority has no dependence on number of jobs in the queue. In fact, an important purpose of fairshare is to guarantee that no one user can dominate the machine. It may *appear* that on user is, but as that user's historical run time accumulates, his fairshare priority will decrease.
- Remember that LSF information is available to all cluster users. Use **bqueues -lr** to see priorities and historical runtime for all users. Do NOT use **bjobs -u all**.
- Another command you can try is **ckpjobs** (or **ckpjobs -v**) but remember that this gives only a momentary snapshot.
- **UPDATE:** QA started using host-based Fairshare on March 22, 2005. Los Alamos, Livermore, and Sandia each share 1/3 of the QA compute nodes. This change allows one Lab to utilize more than 1/3 of the compute nodes if one of the Laboratories is not using it's full allocation at the moment.

Additional Monitoring Tool

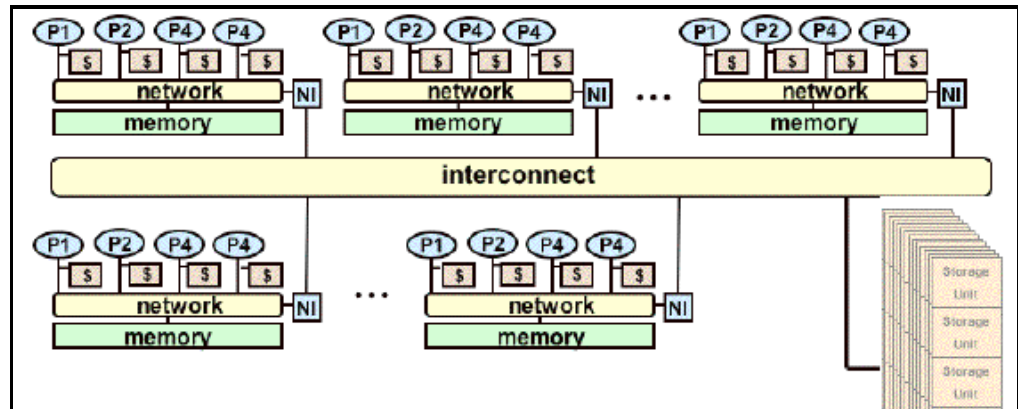
- You can get information about jobs in the queue in semi-graphical form from <http://icnn.lanl.gov/drm/alljobs>. Click on the system you are interested in, and then on the queue you want information about. Jobs are listed with their status, cpu count, submission, and start times (for running jobs), as well as a priority, normalized to one. If the job load remained static, the pending job with the highest priority would run next. A start time is displayed for any jobs that have started reserving processors. Even a job that is reserving processors, though, may go down in priority relative to other jobs.

Q Architecture

- Computer architecture defined:

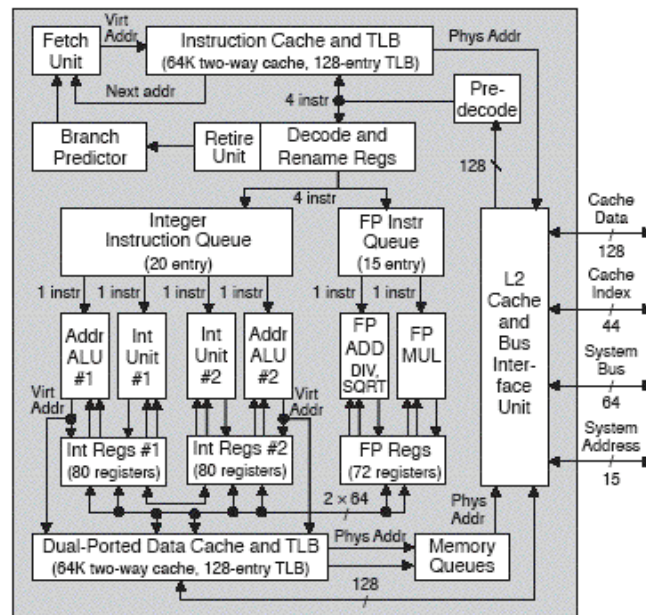
"the attributes of a [computing] system as seen by the programmer, i.e., the conceptual structure and functional behavior, as distinct from the ... logic design and physical implementation." Amdahl, Blaaw, and Brooks, 1964, quoted by D. Patterson,

- The components we will consider include:
 - The Central Processing Unit (CPU), memory subsystem, and node.
 - Parallel architecture, which extends traditional computer architecture with a communication architecture (interconnect hardware, topology, communication processor, & programmer abstraction).
 - The storage subsystem.
 - The ASC Q systems are all a type of hierarchical architecture known as "cluster of SMPs" and a schematic view of this architecture is shown below. Other important SMP clusters include Blue Mountain and ASCI White, although many of the characteristics in these systems are quite different. "SMP" means symmetric multiprocessor, a system in which several processors share a single physical memory via a bus or crossbar switch.



The Alpha Processor

- A schematic view of the 21264 Alpha processor (also known as EV68) is shown below.
- This EV68 is the third generation of the Alpha processor. The first generation, the 21064 or EV4, was introduced in 1992 and was one of the first 64-bit microprocessors. The EV68 used in Q is not the latest version of Alpha; the EV7 system, several notable microarchitectural features of which are described in a [recent paper](#), is the latest (and final) version.



Alpha EV68 Chip Architecture

Six basic parts of this architecture:

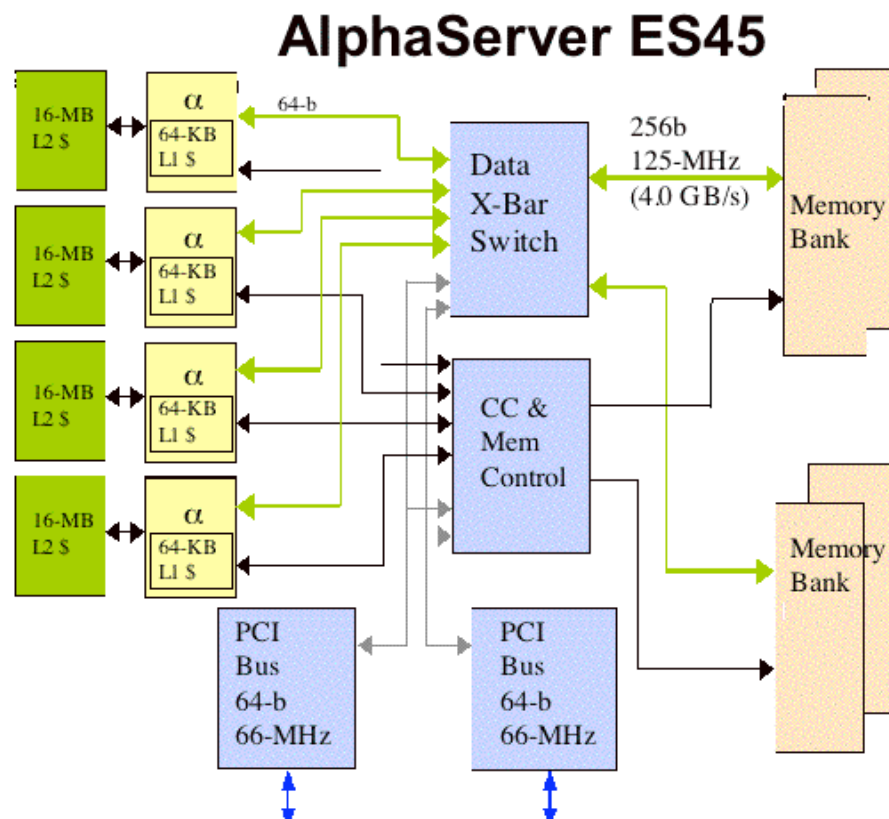
- Instruction fetch, issue, and retire unit (Ibox)
- Integer execution unit (Ebox)
- Floating-point execution unit (Fbox)
- Onchip caches (Icache and Dcache)
- Memory reference unit (Mbox)
- External cache and system interface unit

- The Alpha processor is a full 64-bit system, meaning that all registers are 64 bits in length, all data paths are 64 bits wide, and the processor issues instructions that operate on 64-bit quantities.
- Two floating-point formats are available: IEEE standard and one compatible with the VAX
- The Alpha features out-of-order instruction execution with up to 80 instructions "in flight."
- The Alpha processor is *superscalar*: Able to fetch and issue up to 4 instructions per clock period (CP) and initiate execution on up to six instructions per CP:
 - 2 Integer instructions
 - 2 64-bit loads, 2 64-bit stores, or a load and a store
 - 2 Floating-Point instructions
- Two floating-point instructions per CP implies that the peak processing rate is essentially two times the CPU clock rate ($1.25 \text{ GHz} * 2 = 2.5 \text{ GFLOPS/s}$). However, this is a highly optimistic view - assumes:
 - Ratio of FLOPS to other operations is essentially as above. Rare.
 - FLOPS within a given code sequence are independent. Generally true.
 - Balance of * and +. Generally true.
 - All data available in 1 CP (nearly impossible)

Latency and Repeat Rates for Floating-Point Operations		
Operation	Latency (CP)	Repeat Rate (CP)
Add	4	1
Mult	4	1
64-bit SQRT	33	(not pipelined)
64-bit Divide	15	(not pipelined)

The ES45 Node

- The ES45 server node consists of four Alpha processors, each with a 16-MB external Level-2 data cache, a chipset containing the memory controller and I/O subsystem connections, and (in ASCII Q) up to 32 GB of memory uniformly accessible by all four processors in the node.



- The architectural feature of a system like the ES45 that most contributes to user code performance is the memory hierarchy. Two memory-related performance metrics are latency and bandwidth. Measured memory latencies for each level of the memory hierarchy in the ES45 are listed in the following table. Memory latency is the time that elapses between when the processor issues a memory request and when the first datum for that request arrives at the processor.

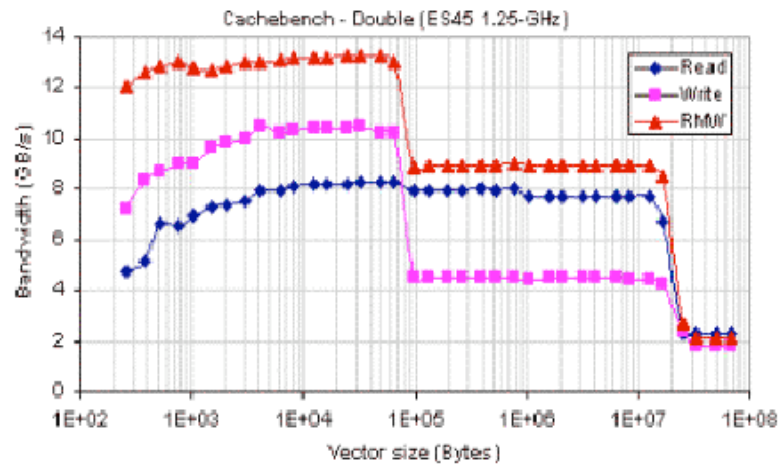
Cache Characteristics for the ES45			
	Latency (in CP)	Cache Size	Line Size
L1 Cache	2	64 KB	64 Bytes
L2 Cache	19	16 MB	64 Bytes
Main Memory	144	8-32 GB	

- A key feature of the ES45 is its **Uniform Memory Access**, meaning that the memory latency listed above is the same for all of the (four) processors in the node. The tradeoff, of course, is that there are relatively few processors per node. In contrast, the computing node used in ASCI Blue Mountain (known as the sgi Origin2000) has 128 processors but each processor can experience a 3-fold difference in memory latency, depending on how the OS allocates memory. The memory system in the Origin2000 is known as a NUMA (Non-Uniform Memory Access) system.
- For reference, the next table shows how memory latencies have grown in the 3 generations of Alpha microprocessors. This table shows why memory latency is such an important factor in user code performance and why peak processor performance is not achieved. Peak performance assumes that all data are immediately available to the processor (i.e., latency of 1 clock period or less).

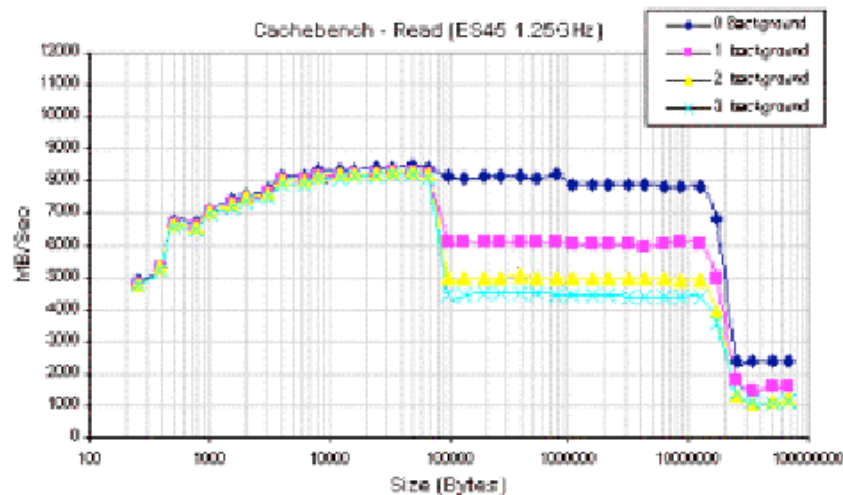
Memory Latencies in 3 Generations of Alpha Microprocessors			
	21064	21164	21264
Latency (ns)	340	266	115
CPU Clock period (ns)	5	3.3	0.8
Ratio	68	80	144
CPU Issue Rate	2	4	4
# Instructions	136	320	576

- These data, taken in part from "Computer Architecture A Quantitative Approach," by Hennessy and Patterson (Morgan Kaufman, San Francisco, 3rd Edition, 2003), show how much computation is potentially lost while the processor stalls waiting for memory; or alternatively, how much computational work you would have to do in parallel to the memory accesses in order to try to entirely hide the effect of the memory latency.
- Memory bandwidth measurements from the 21264 are shown in the graphs below and are summarized in the table.

Measured Memory Bandwidth in the ES45	
L1 Cache	7.9 GB/s
L2 Cache	7.5 GB/s
Main	2.2 GB/s



- Keep in mind that the memory bandwidth available to an ES45 processor can decrease depending on the number of processors simultaneously accessing memory (a decrease of up to a factor of 2 is possible). The next graph shows this.



- Which of the two (memory latency or bandwidth) affects your code more depends on the dominant memory access pattern.

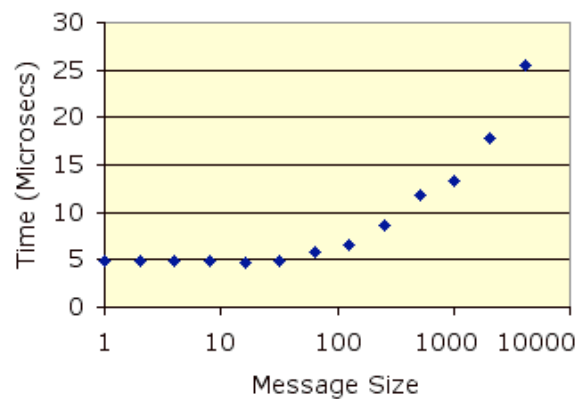
The Quadrics Interconnect

- The Quadrics system area network for ASCI Q consists of two primary components:
 - The **Elan** Adapter Card is a communications processor packaged on a PCI-based network adapter card. This card provides the interface between the ES45 node and a multi-stage network. Copper cabling connects each Elan card to a switch.
 - The Elan card contains a 32-bit RISC processor used to offload processing related to message passing from the ES45 Alpha CPUs.
 - **Elite Switch:** 8-way, bidirectional crossbar switch. Each port connects to either an Elan adapter card or to another switch. Elites are packaged on switch cards with various configurations.
 - [Images available here.](#)
- The measured latency for a unidirectional "ping" on ASCI Q is about 5 microseconds. Under bidirectional traffic, in which two MPI processes placed on adjacent ES-45 nodes ping each other, the latency is slightly higher (about 7 micros).
- The asymptotic bandwidth for unidirectional ping is 293 MB/sec which is the peak bandwidth that can be delivered by the PCI bus of the ES-45. When both nodes ping each other the observed asymptotic bandwidth is about 130 MB/s. This represents about 90% of the nominal

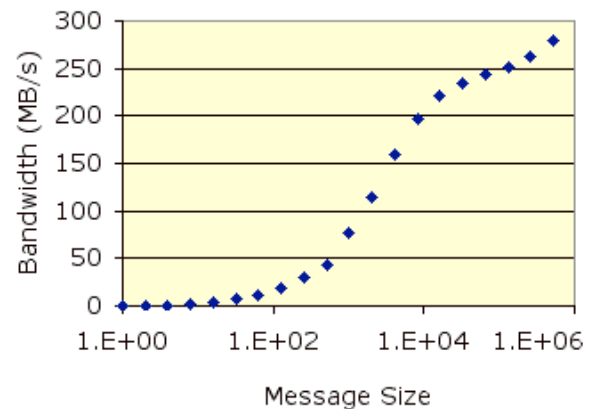
value.

- The ES45 has two 66-MHz PCI bus slots. In ASCII Q we plug an Elan card into both of them and construct two separate Elan/Elite networks - called a **dual-rail** system.

Quadrics Ping Latency

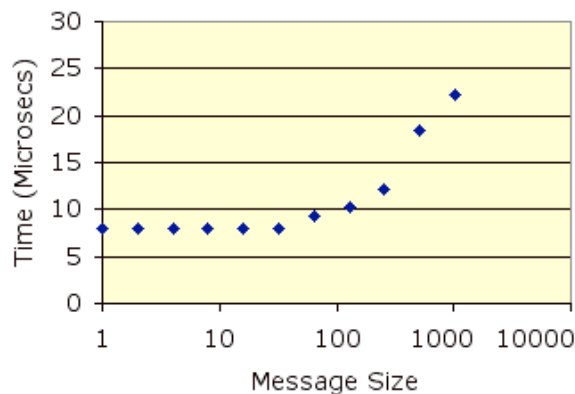


Quadrics Ping Bandwidth

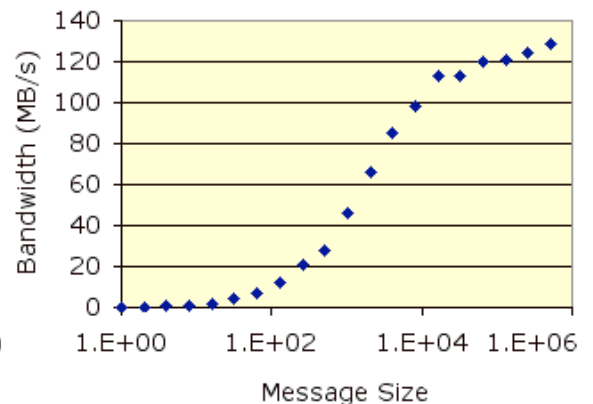


- When two Elans are present within the node the asymptotic bandwidth increases by approximately 180% if simultaneous messages can take advantage of the two rails. Individual messages are not striped across rails.

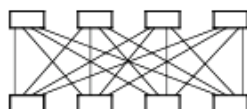
Quadrics Dping Latency

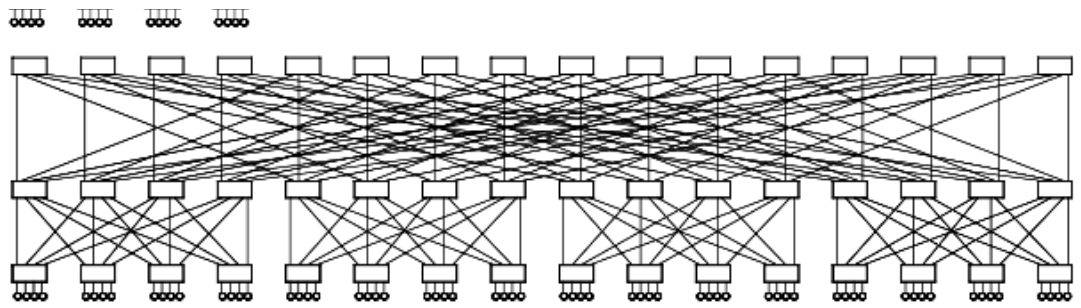


Quadrics Dping Bandwidth



- Can use one rail to send and the other to receive. The dual-rail configuration can increase overall network bandwidth available to a node but the application has to be able to use it.
- Dual-rail could potentially also increase overall system availability. However, rails are statically allocated; therefore, if a rail crashes mid-job, the job with the process that was using that rail will crash. (In practice, the network rarely crashes.)
- The Quadrics network on the Q systems is also used for global file system sharing across the cluster.
- An important advantage of the fat tree topology used in the Quadrics network is hardware support for collective communication patterns, making operations such as broadcast and barrier fast and scalable.
- Multiple Elite switches are combined so that the network has a quaternary fat-tree topology. Example fat-tree networks of dimension 2 and 3 are shown in following figure. A quaternary fat-tree of dimension n connects 4^n processing nodes using $4^n - 1$ switches.





- Each Elite switch contains an internal 16x8 full crossbar. In the ASCI Q systems the Elite switches are packaged in 128-way boxes.

The first level of boxes (called "[node-level switches](#)") implements the first 3 levels of the fat-tree. Each of the 16 node-level switches uses 64 down and 64 up ports to interconnect 64 nodes to the next-highest level of switches.

The second level of boxes (called "top-level switches") implements the remaining 2 levels of the fat-tree required for a system of 1024 nodes. There are 16 top-level switches in the 1,024-node systems. The top-level switches also provide up links to "super top switches" which are capable of interconnecting QA and QB into a single 2,048-node system.

QSC is a 256-way, dual rail cluster that uses 8 QSW 64U/64D Elite-3 node-level switches, four per rail and 8 QSW 512-way Elite-3 federated switches, four per rail.

Storage Systems

- **Local Disk Filesystems** Each AlphaServer ES45 has at least one locally-attached disk drive on which `/tmp` and `/local` are mounted. This internal storage should be considered as highly volatile and fragile! In particular, there is no RAID protection, no backup of any kind, and LSF wipes this space at the beginning and end of every LSF job. Additionally, if this space fills, the node and or certain system software (such as the compiler) will crash.
- **Cluster File System** Recall that, for ease of management, each Tru64 system is organized into multiple 32-node domains.

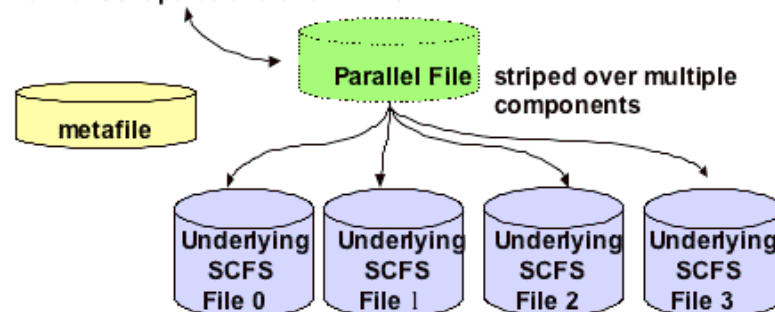
Each domain, which is called a Cluster File System (CFS) domain, has a single root filesystem with the OS stored on fiber channel storage attached to the first two nodes in the domain. These nodes export this filesystem to other nodes in the domain through Quadrics.

- **Parallel File System** A separate Storage Area Network (SAN) is used to provide RAID-based global user storage in the Q systems. The filesystem mounted on these SAN devices is the Parallel File System (PFS). The `/scratch1` and `/scratch2` (and some other spaces) are PFS filesystems. They are primarily for large-scale, high-performance, parallel I/O.

Parallel File System (PFS)

- Two basic things you need to know about the Tru64 PFS systems:
 1. PFS systems appear to the user as cluster-global filesystems but they actually formed by the aggregation of a large number of individual *component filesystems*.

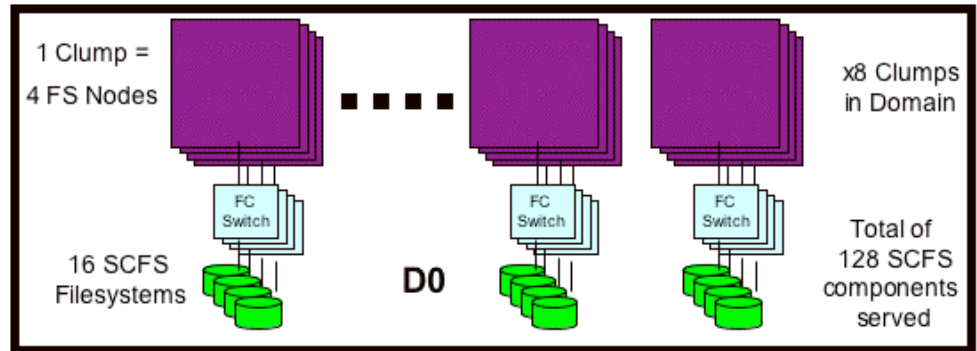
Normal I/O Operations and MPI-IO



The component filesystems consist of special ES45 nodes, called **file server nodes** that are physically attached to the storage units (RAID disk systems) and are serve their component file systems, via an HP AlphaServer SC high-speed proprietary protocol (**SCFS**), to all other domains in the cluster. SCFS is basically NFS over Quadrics.

In principle, these fileserver nodes could be anywhere in the cluster. On QA, QB, and QSC we collect them into "file server domains." The fileserver domains serve I/O requests for all of the rest of the cluster. On QB the fileserver domains are qd32, qd40, qd48, and qd56. On QSC they are qscd0 and qscd7.

To increase reliability, no storage unit is attached to only one file server. In fact, each storage unit is attached to four. We call the four-file-server-storage unit a "clump." Each clump contains 16 SCFS component filesystems.



Sample configuration: QSC

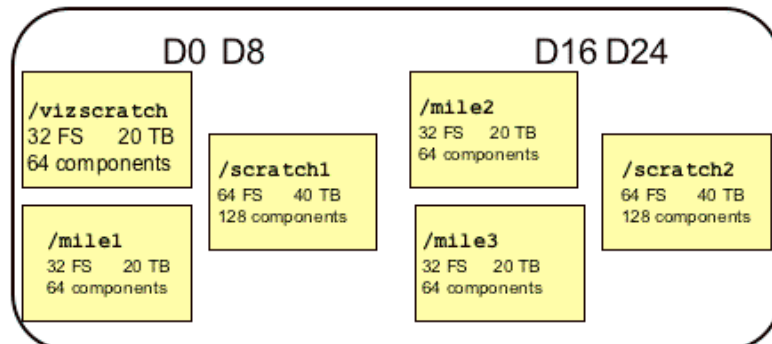
- o 256 total nodes in 8 domains
- o 64 AlphaServer ES45 file server nodes in 2 FS domains
- o User processes run on the file server domains
- o Total of 30 TB PFS scratch in two filesystems (/scratch1 and /scratch2) with 64 components, each.

Sample configuration: ASCI QA

- o 1,024 total nodes in 32 TruCluster domains
- o 896 compute server nodes in 28 domains
- o 128 file server nodes in 4 file server domains (D0, D8, D16, and D24 on QA) (D32, D40, D48, D56 on QB)
- o Each file server domain serves 128 SCFS component filesystems to the rest of the cluster - 512 total PFS components.
- o So there are 4 SCFS systems per file server node. A **clump** consists of 4 file server nodes and a storage cabinet interconnected via Fibre Channel. (On QA and QB, that is. On the C clusters there are 2 nodes per storage cabinet.)
- o Each SCFS component has about a 1-TB capacity.
- o 160 TB total global SAN storage on each of QA and QB.

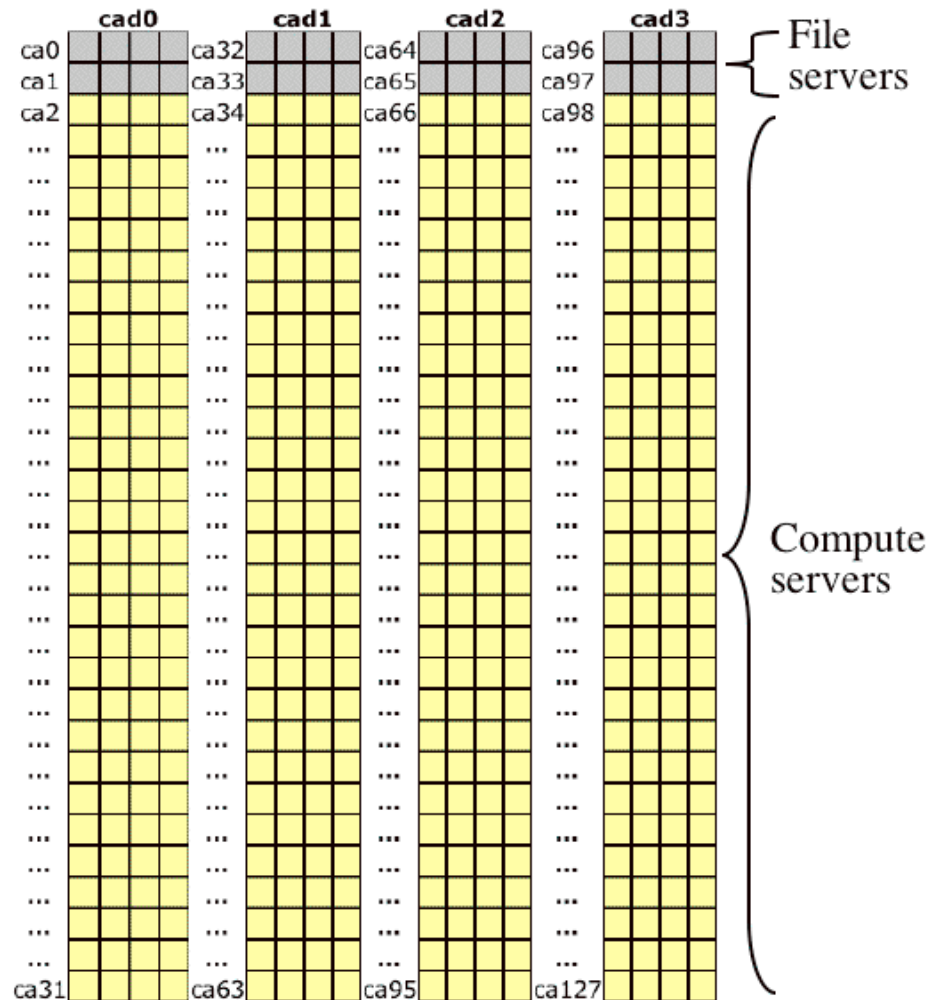
QA PFS

4 FileServer domains 128 FS nodes 512 SCFS components



On the C systems (CA, CB, CC), the first two nodes in each domain are the file servers, serving files to the rest of the nodes in their respective CFS domains, e.g., on CA, CB, and CC, each domain has

- 2 nodes reserved for file services, and
- 30 nodes (120 processors) available for computing.
- (No user jobs run on the 2 file server nodes.)
- Example: CA



CX is different because it has only a single domain, called "cxd0." As you know, this domain consists of 32 nodes (128 processors). CX has no global parallel file system, and therefore, no filesystems, so all 32 nodes (128 processors) are available for computing.

- I/O on the CX cluster is optimized for serial I/O, not for parallel I/O.

2. Data written to PFS filesystems can be striped across multiple components.

For data striping the relevant PFS attributes are:

- **NumFS:** Number of component file systems. Configured at boot time.
- **Stride:** Amount of data transferred to/from a single component before the next component is used. Default value set for PFS but can be changed by the user.

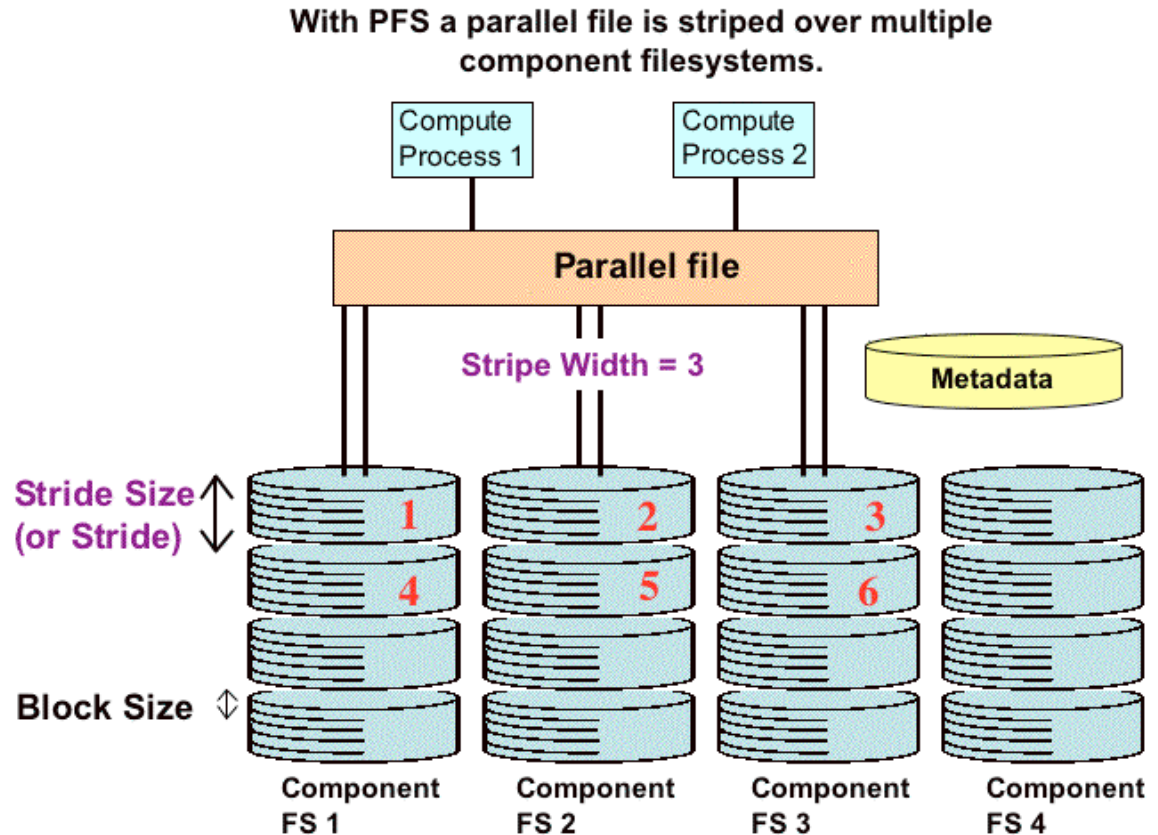
Stripe: Number of component file systems to stripe data across. Default value set for PFS but can be changed by the user.

Block size: Amount of data transferred in a single I/O operation. Configured at boot time.

Default configuration:

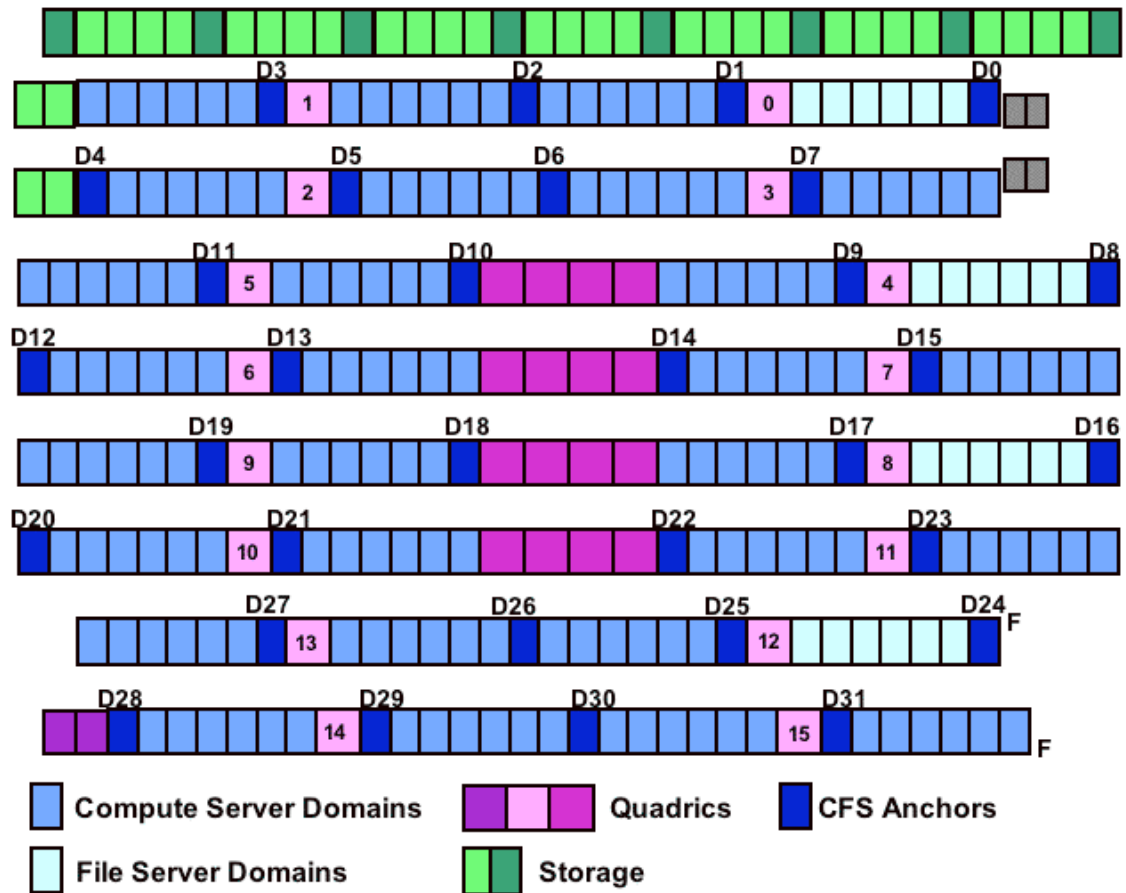
- Stripe = 1 component per file
- Stride = 8 MB per stripe
- Block size = 64 KB per operation

When a file is created, the first component file system that data is written on is chosen at random.



The file system has a default stride and stripe, but the user may find it desirable to override the defaults and set these parameters on a per file basis. The optimal settings for stripe and stride are highly application dependent (see below).

- For the big LANL Q systems (QA and QB) the file server nodes also run normal user jobs (albeit only interactive jobs). Not true on the C clusters.
 - AlphaServer systems do not need to run PFS, and in fact, one ASCI Q cluster at LANL does not: CX. I/O on CX is optimized for large, serial data transfers, not parallel I/O. So all I/O on CX is handled by CFS and its underlying filesystem, AdvFS.
-
- Now we can look at the layout of ASCI Q. To understand why the file server domains are distributed throughout the cluster as opposed to being contiguously located, see the [LANL paper](#) on this subject.



[Contents](#)

Best I/O Practices on Q

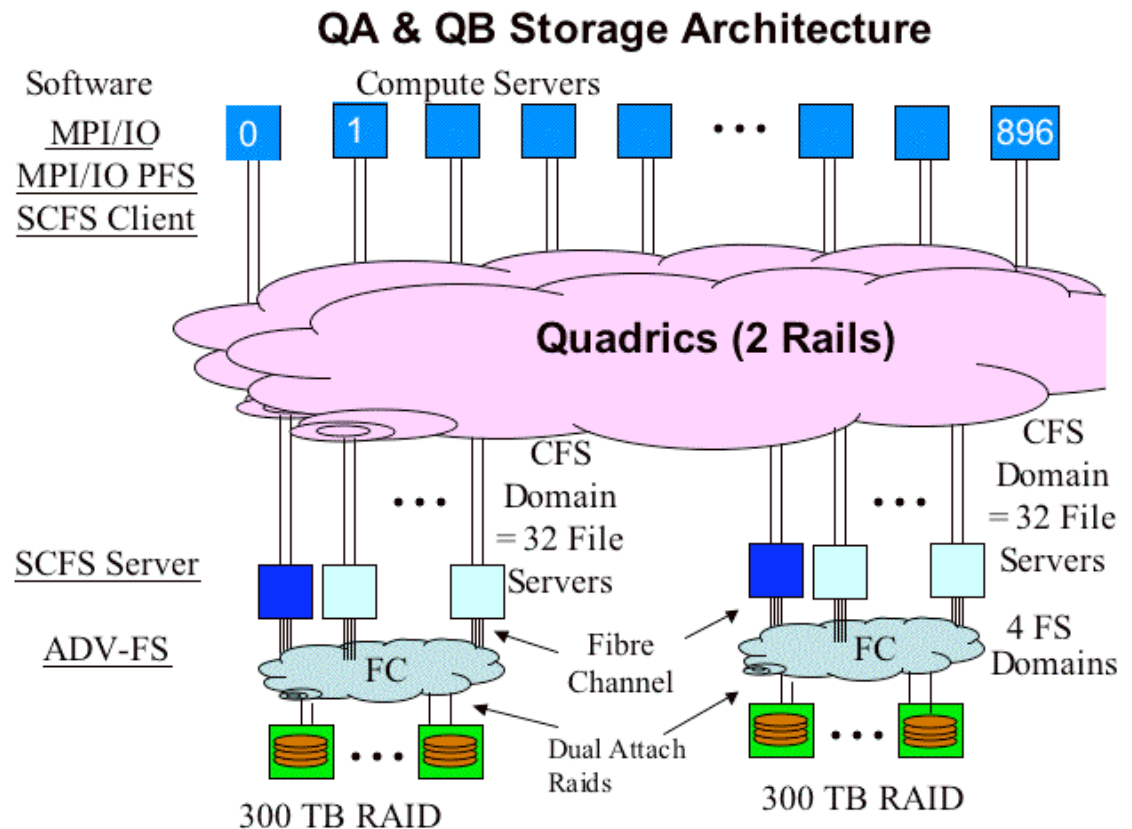
- In this section we attempt to provide information on how to get the best I/O performance on Q systems, information ranging from high-level, relatively simple tips that can be used by anyone to low-level, detailed issues that may require code and/or programming model changes.
- This section is a modified version of a presentation given by members of the LANL CCN-8 [Run Time Systems Team](#) (LANL Yellow partition only) based on extensive experimentation they have done on Tru64-based systems. Members of this team used to include Gary Grider (Team Leader), Hsing-Bung Chen, Harry McGavran, Marydell Nochumson, James Nunez, and Julie Stidham .

You can see view graphs from the original presentation on <http://computing.lanl.gov/article/397.html>.

- Review: Q's storage architecture in a nutshell:
 - An HP AlphaServer SC systems is constructed through the tight coupling of 4-processor AlphaServer ES45 nodes.
 - For ease of management, the nodes are organized into 32-node Cluster File Systems (CFS) domains which share a common root filesystem. Two nodes in each CFS domain serve as "anchors" - they are attached to RAID disks containing the root filesystem and to the external LAN.
 - Certain ES45 nodes are attached to RAID storage devices and these nodes serve the global filesystems to all other nodes via HP's SCFS protocol. Elan data transfers can be

overlapped with disk I/O.

- The PFS filesystems are constructed from multiple SCFS components, over which data can be striped.



1. Remember the difference between PFS and NFS filesystems and that there is no uniform "best" solution; there are only tradeoffs between accessibility, space, performance, fragility, etc..
 - PFS: `/scratch1`, `/scratch2`, `/mile1`, `/mile2`, `/mile3`, `/vizscratch`. Use for very large files, for parallel I/O. Do not use for tar, compiling/linking, small I/O, especially serial, or lots of formatted reads/writes. Do not put executables here. Do not use as part of `LD_LIBRARY_PATH`.
 - NFS: `/netscratch`, `~`, `/usr/projects`, `scratch` (on CX). Use for sequential, small-operation I/O.
2. Never set the execute bit on the permissions on a data file on PFS, because PFS will revert to NFS.
3. Try to use reasonable size I/O operations, at least a few hundred kilobytes.
4. Never put over 10,000 files in a single directory on any file system.
 - File management operations require `stat()` on every file, which uses a linear linked-list data structure so that the time to do it grows *worse* than linearly with number of files. Expect 5 ms per file (25 seconds for 5,000 files!); even slower per file for larger directories. This is true for both PFS and NFS.
 - If you absolutely must have $O(5000-10,000)$ files in one directory, studies by Nunez, et. al (CCN-8/9), have shown that under such conditions NFS is faster than PFS on the Q systems for operations such as file creation and `stat` operations such as `ls -al` and `find`.
5. HP fortran has unbuffered i/o as a default. By adding either `"-assume buffered_io"` to the compiler options, or `"BUFFERED='YES'"` to the particular Fortran `OPEN` statement, the code will then buffer output in memory, and only write it to disk in large chunks. Of course, this

only helps with writes, primarily on PFS systems, and only with small (< 1 MB) writes.

6. I/O rates on all systems may vary considerably from one run to the next. LSF job placement may be a cause. On a PFS, and especially if very small reads and writes are involved, it may help to run the code on a fileserver domain that serves the components of that PFS, because in that case, the PFS components are accessed as a local, fiber-channel AdvFS filesystem. If the PFS has to be accessed remotely via SCFS the job may run slower. This is particularly true on QSC.
7. Contact the LANL consultants (5-4444; e-mail: consult@lanl.gov) to help you resolve problems. They track your problems and bring in appropriate help as needed.
8. Do not write very large (~500 GB) sequential files on a PFS with a stripe-width of 1.
 - On a PFS file system on Q, using large (**O**(100's of KB), sequential I/O operations you might expect about 53 MB/sec. The same operation writing to an NFS filesystem on Q might get about 75 MB/s. However, data rate is not the issue. Space is the issue.
 - PFS is essentially *smaller* than the sum of its parts because total free space is calculated by multiplying the number of components by the smallest free space on any component. Thus, a four 4-GB-component PFS system in which one component has only 2 GB of space available is 8 GB (not 16 GB).
 - If a single component of a PFS fills the entire PFS will appear full. Files on PFS have a default width of 1. If a file is going to be very large, you should set the file's width to be greater than 1. A good rule of thumb would be to not put more than 1-2 GBytes per component. You can set the width of a file before writing to the file using the **tpf** (parallel touch) command.

tpf default /full/path/PFS/directory Returns system defaults for stripe, stride, etc.

tpf get /full/path/PFS/file Returns the PFS parameters for a file

tpf set /full/path -stripe # -stride # -start_iodevice # -force

Creates a file with specific PFS parameters. Notes:

- **#** is an integer.
- All flags are optional but if a parameter is not specified then the default value will be used.
- Stripe can be between 1 and N, where N is the number of PFS components.
- Stride must be at least 64 KB.
- **start_iodevice** is between 0 and M, where M is the number of components in the SCFS - 1.
- The **-force** flag is used to impose new PFS parameters for an existing file.
- NOTE: if you use the **tpf** command to change PFS parameters for an existing file that file's contents will be wiped out. In other words, if you want a non-default stripe size you must set it before you write anything to the file.



Sample Output Using tpf Command

```
qsc0% tpf get IO_test.bin.000
PFS parameters for file IO_test.bin.000:
Base I/O device: 25
PFS Stripe: 1
PFS Stride (bytes): 8388608

qsc0% tpf set IO_test.bin.000 -stripe 3
File IO_test.bin.000 exists and the force flag is not on.
File IO_test.bin.000 was not modified.

qsc0% tpf set IO_test.bin.000 -stripe 3 -force
File set PFS attributes set to:
Base IO device (pfsmmap_slice.ps_base): 25
PFS Stripe (pfsmmap_slice.ps_count): 3
PFS Stride (pfsmmap_stride): 8388608

qsc0% rm IO_test.bin.000
qsc0% tpf set IO_test.bin.000 -stripe 2
qsc0% ls -al IO_test.bin.000
drwx-----  3 hjw      psem  8192 Apr 19 08:17 .
drwxrwxrwx 581 root    system 16384 Apr 14 12:15 ..
-rw-----  1 hjw      psem    0 Apr 19 08:17 IO_test.bin.000

qsc0% tpf get IO_test.bin.000
PFS parameters for file IO_test.bin.000:
Base I/O device: 35
PFS Stripe: 2
PFS Stride (bytes): 8388608
```

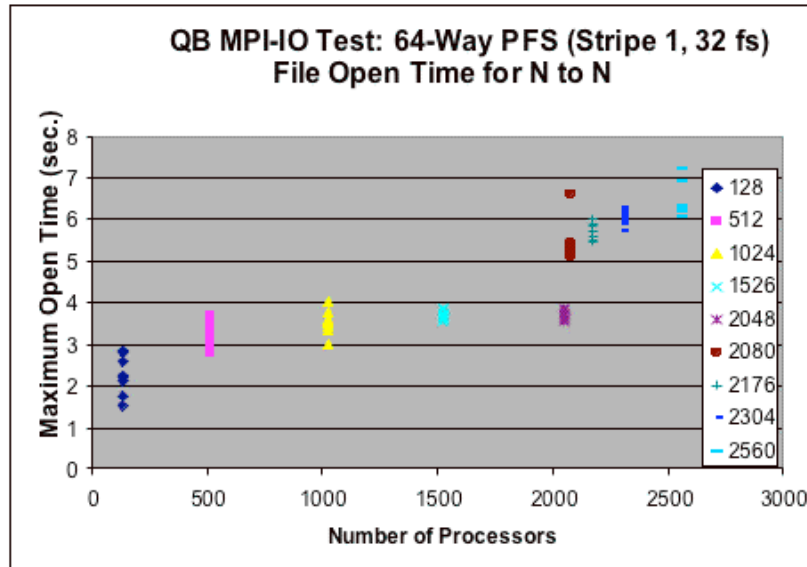
9. Certain kinds of operations can run much faster from the `/tmp` filesystem than from a PFS or NFS filesystem (particularly PFS). Examples include `tar`, `make`, (i.e., compiling), and `configure`.

This space exists on the local disk drive on the compute nodes and writes to it are generally cached in the node's memory so they go very quickly.

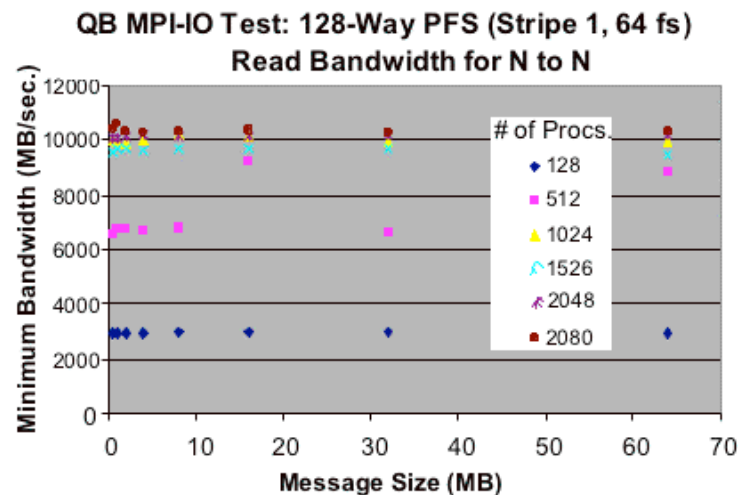
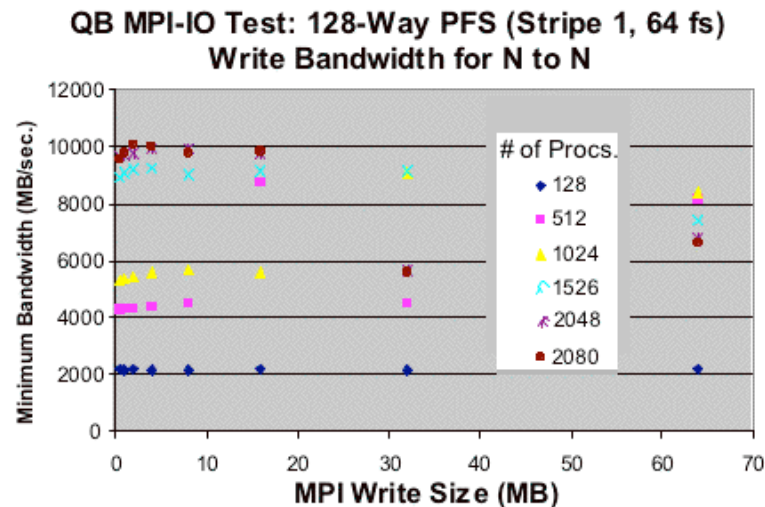
Caution: All files in `/local` are ELIMINATED at the end of your LSF job! And if the `/local` file system fills up, the node will crash.

Of course, you incur the overhead of copying in and copying out of `/tmp` but in many cases this will be smaller than the time to run in a PFS filesystem.

10. Another consideration if you absolutely must have $O(5000-10,000)$ files in one directory and you need to find a file or a few files in a huge directory, use the locally-developed `mpifind` command. Contact the consultants if this is of interest to you.
11. For parallel workloads it is highly recommended that you use a higher-level I/O library, such as UDF, HDF, or MPI/IO.
12. Parallel N-to-N or N-to-M Where M is Large
 - This means "N" compute processes writing to/reading from "N" or "M" separate files.
 - If N is large, use PFS for this. PFS scales well whereas NFS does not.
 - Use large block sizes (100s of Kilobytes and up).
 - If N is large, there is no need to use large file width, as each file will start on a different disk randomly, so using the default file width of 1 is reasonable.
 - As you can see from the following graph, overhead to get the files open is quite small and seems to be quite parallel if width is narrow as it should be when N is large.



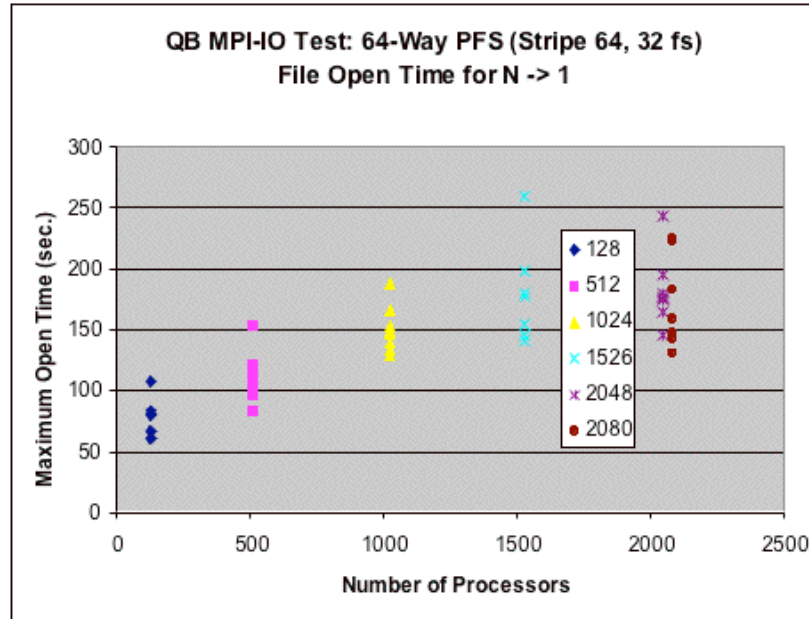
- As you can see from the following graphs (from James Nunez), aggregate data rates are quite good over a wide range of read/write sizes except for very small operations - 50-60 MB/second per process (which will degrade to 100 MB/second per compute node) up to a maximum PFS performance of 60-100 MB/second per component. Performance can also vary from one filesystem to another.



- Remember that if you create large numbers of files and/or very large directories, file management operations may become very slow. (see above.)

13. Parallel N-to-1 Write

- Data rates for N processes writing a single file are not quite as good as in the N to N case, but can still be respectable. There are two issues: open overhead and stripe width.
- Data rates for N processes writing a single file can be dominated by file open times, which are much higher than for the N to N case and grow with the number of processors (see graph).



- The file open overhead can be reduced to 5-20 seconds using two locally-introduced open methods but you must use MPI-IO for these. The two methods are the "Serialized" method and the "Overlapped" method. Contact the ICN Consultants if you need to use these methods.
- For N to 1 operations there is also a stripe width effect. Data rate is directly proportional to the stripe width set for the file assuming the number of processors is reasonably large.
 - In parallel, each stripe can write at up to 30 MB/sec, so a stripe of one yields 30 MB/sec, stripe of sixty yields 1-2 GB/sec, etc., up to a max stripe size for the file system. **So setting a large stripe width for N-to-1 file operations is very important!**
- Reminder: You set the stripe width either with MPI-IO file hints, or with `ioctl` calls, or with the LANL `tpf` utility from the shell.
- Here's how you control stripe width and stride using MPI-IO in C:

```
MPI_Info info;
MPI_Info info = MPI_INFO_NULL;
...
MPI_Info_create(&info);
MPI_Info_set(&info, "striping_factor", "2");
MPI_Info_set(&info, "striping_unit", "8388608");
MPI_Info_set(&info, "start_iodevice", "3");
MPI_File_open(MPI_COMM_SELF, tfname, MPI_MODE_CREATE|MPI_MODE_WRONLY,
              info, &wfh);
```

- In the above example, `striping_unit` is the stride (set to 8 MB). The stripe width is set to 2. Note the use of string values.

- Here's how you control stripe width and stride using POSIX I/O in C:

```
struct pfsmap my_pfsmap;

fd = open(fname, O_RDWR|O_CREAT);
my_pfsmap.pfsmap_slice.ps_base=3;
my_pfsmap.pfsmap_slice.ps_count=2;
my_pfsmap.pfsmap_stride=8388608;
ioctl(fd, PFSIO_SETMAP, &my_pfsmap);
close(fd);
```

- The above example creates a file with a stripe width of 2, starting disk number 3, and with stride 8 MB.
- Note: Unlike MPI-IO, when you set the file parameters with this call, you must set a number for all three parameters or it will default to the very undesirable value of zero.
- Remember: All values must be set before any data are written to the file!!!

```
ioctl(fd, PFSIO_GETMAP, &my_pfsmap)
```

14. Using the **pcp** Parallel Copy command

- The parallel copy command (pcp) takes advantage of the fast Quadrics interconnect on the Q systems by using MPI to copy a file in parallel.
- If copying a file to a PFS system you can set the stripe width, stride, and starting disk.
- Performance comparison to copy a 10-GB file from and to /scratch2 on QSC during normal use (with other users on the system):

```
cp                      480 sec, 21 MB/s
pcp with 8 processors  50 sec, 202 MB/s
```

- On QSC, CA, CB, CC, QA, and QB, **pcp** is located in: **/usr/local/share/etc** and the executable is **pcp.x**.
- You must load the MPI_default modulefile before using it.
- **pcp** usage: **pcp.x in_fname out_fname nReadProcs nWriteProcs striping_factor start_iodevice**

15. Using the **kpc** Kerberized Parallel Copy command

- Use this command to move large multi gigabyte files between LANL computing platforms such as Blue Mountain, and the multiple segments of Q. Enormous data files may need to be moved at high data rates.
- Results:
 - QB --> b23: 387 MB/sec
 - QB <-- b23: 239 MB/sec
 - QB --> QA: 516 MB/sec
 - QB <-- QA: 618 MB/sec
- Must module load an MPI modulefile.
- Location: **/usr/local/packages/kpc/<MPI_Version>/<kpcget><kpcput>**
- Usage: **kpc [-d] [-x] [-stripe N] [-gigs M] blocksize infilename outfilename rnode1 [rnode2] ...**

16. HPSS Performance Hints

In addition to getting good I/O performance within your application, you also want to make sure that you get reasonable performance when you transfer data to or from HPSS. This is particularly true if you've generated large parallel files on PFS. Since the Q systems are now

undergoing regular maintenance of their PFS file systems, which means unavailability for some time, you have to be able to rapidly store to HPSS.

This section, which consists of five "hints" for obtaining good HPSS performance, was developed by the LANL CCN-7 [Storage Systems Team](#), particularly Mark Roshke and John Blaylock.

We assume here that you already know how to use HPSS via the psi utility. If not, you can view the man page on <http://int.lanl.gov/hpss/PSI.html> or you can view a tutorial on psi on <http://asci-training.lanl.gov/BlueMountain/HPSS.html>.

Please keep in mind the central idea of this entire tutorial, namely, that there are tradeoffs in how you do your work. Do not view any of these hints in isolation, in particular, make sure you understand how they relate to the concepts mentioned in the sections above.

1. On worker machines, request multiple nodes (up to 16) from LSF when large files or many small files are to be transferred. For files larger than approximately 100 MB, typical transfer rates are 40-50 MB/sec per node, and overall performance should be roughly proportional to the number of nodes.
[View more info.](#)
2. On the Q machines, large files on the parallel file systems will generally achieve greater transfer rates to HPSS if they are striped across multiple components. Observed transfer rates should be approximately 40-50 MB/sec per component, provided PSI has enough nodes available.
[View more info.](#)
3. Avoid transferring large numbers of small files (less than approximately 10 MB) to HPSS. Since there is an overhead associated with each file transferred, one large file will transfer much faster than many small files. When the generation of lots of small files is unavoidable "tar" up small files and store the "tar" file to HPSS. Also, HPSS actually does better when you just give it a file tree to transfer.
[View more info.](#)
4. Do not run multiple PSI commands to improve performance - add more nodes instead. Multiple PSIs can actually reduce overall performance due to excessive resource contention.
[View more info.](#)
5. If you are processing many files (>1000), avoid the use of the `-cond` option in psi; it can reduce performance.
[View more info.](#)

[Contents](#)

Performance Analysis on Q

Timing Your Code

- Here is a relatively low-overhead timer that is portable across most Unix systems.

```
#include <sys/time.h>
#include <sys/times.h>
#include <sys/resource.h>

void
timers(cpu,et)
double *cpu,*et;
{
    struct rusage r;
    struct timeval t;

    getrusage( RUSAGE_SELF, &r );
    *cpu = r.ru_utime.tv_sec + r.ru_utime.tv_usec*1.0e-6;
```

```

        gettimeofday( &t, (struct timezone *)0 );
        *et = t.tv_sec + t.tv_usec*1.0e-6;
    }

```

/* FORTRAN calling flavors */

```

void
TIMERS(cpu,et)
double *cpu,*et;
{
    timers(cpu,et);
}

```

```

void
timers_(cpu,et)
double *cpu,*et;
{
    timers(cpu,et);
}

```

Use it in the following way (from C):

```

(void) timers ( &cpu0, &et0 );
...code to be timed...
(void) timers ( &cpu1, &et1 );
print*,'cpu time for code was ', cpu1-cpu0
print*,'elapsed time for code was ',et1-et0

```

From FORTRAN just use `call timers (cpu0, et0)`, etc..

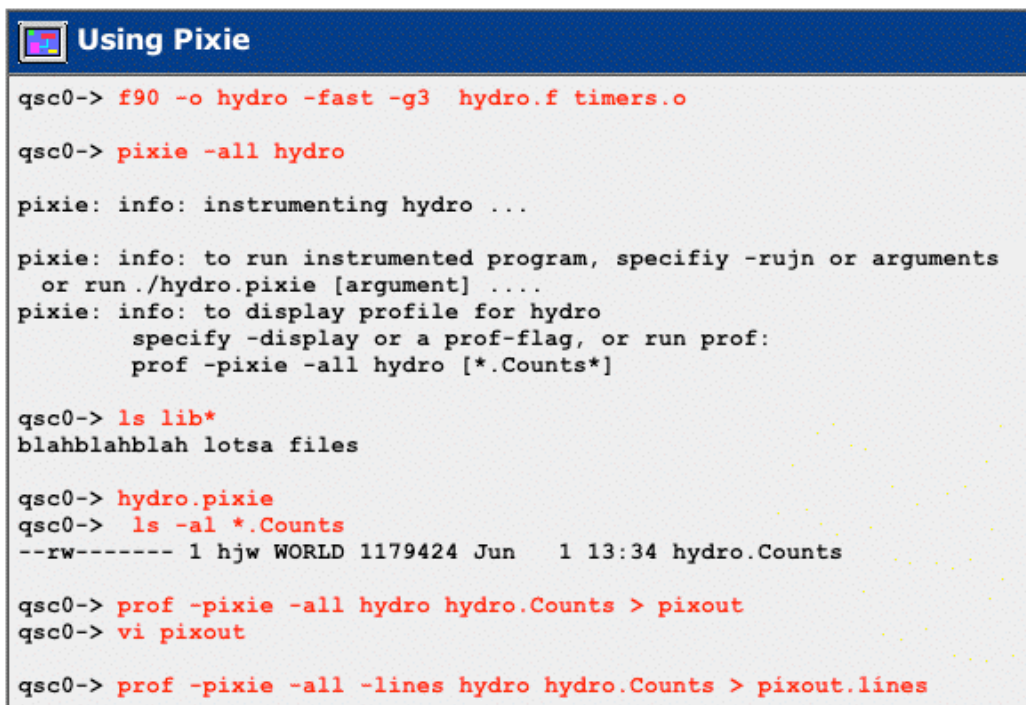
- For a multi-process MPI code, you can also use the MPI `MPI_Wtime` function. It uses a timer on the Elan card and has nanosecond accuracy. Use is similar to the function above: call once before and once after code to be timed and subtract.
- See the [performance analysis article](#) on computing.lanl.gov for more info.

Using gprof to Profile Your Code

- Make sure you compile with `-pg` and `-g3`. Then execute the code and a file `gmon.out` will be in your directory. Then execute `gprof < whatever_the_executable_was_called >`. Usually you want to redirect stdout to a file.
- This only gives a routine-level profile and it is not very useful for multiprocessor jobs.
- Very little overhead of the profiler.

Using Pixie to Profile Your Code

- Rather significant overhead.



```

qsc0-> f90 -o hydro -fast -g3 hydro.f timers.o

qsc0-> pixie -all hydro

pixie: info: instrumenting hydro ...

pixie: info: to run instrumented program, specify -rujn or arguments
or run ./hydro.pixie [argument] ....
pixie: info: to display profile for hydro
specify -display or a prof-flag, or run prof:
prof -pixie -all hydro [*.Counts*]

qsc0-> ls lib*
blahblahblah lotsa files

qsc0-> hydro.pixie
qsc0-> ls -al *.Counts
--rw----- 1 hjw WORLD 1179424 Jun  1 13:34 hydro.Counts

qsc0-> prof -pixie -all hydro hydro.Counts > pixout
qsc0-> vi pixout

qsc0-> prof -pixie -all -lines hydro hydro.Counts > pixout.lines

```

- An example of the output is [here](#).
- Can also be done on multiprocess runs (but probably not recommended).
 1. `pixie -pids a.out`
 2. `prun -n 4 a.out.pixie`
 3. After you run the code you get 4 `a.out.Counts.<PID>` files. You can `prof` any one of them or `prof -pixie a.out.Counts.*`, which will sum the counts from the individual files.
- Should also work similarly on OpenMP multiprocess runs.

Using hipprof to Profile Your Code

- Gives basically same info as a pixie profile but with some possible additional hierarchy info (who called who and how much time spent in the callees).
- `hipprof a.out` (produces `a.out.hipprof` instrumented executable.
`prun -n 1 a.out.hipprof` (produces `a.out.hiout` data file
`gprof a.out.hipprof a.out.hiout > output`
- A full single-processor example is [here](#).

Using uprofile to Profile Your Code

- Uses the on-chip performance counters - mostly useful for L2 cache misses.
- `uprofile bcachemisses a.out`
`a.out` (produces `umon.out` data file
`prof a.out umon.out` OR
`prof -lines -p a.out umon.out`
- Can also get instructions retired, CP, and replay traps, although must be obtained in separate runs.

Miscellaneous Performance Tools

- First step in getting good performance out of a large parallel supercomputer: Make sure you're getting good performance from a single processor.
- [Vampir](#) is a set of analysis tools for parallel code, both MPI and OpenMP. It consists of compilers (C/C++ and FORTRAN), an instrumentation library, and a graphical user interface. VGV provides performance measurement, visualization, analysis, and execution-performance improvement tools for OpenMP and/or MPI applications. you're getting good performance from a single processor.
 - See the article on computing.lanl.gov about it.
 - Using Vampir you can get views of your program like [this](#).
- DCPI and HPCToolkit: See the section on 3rd-party software on computing.lanl.gov (under "shared topics"). Compaq's (formerly Digital) Continuous Profiling Infrastructure (DCPI) permits continuous low-overhead profiling of user programs. The system is efficient enough that it can be left running all the time, allowing it to be used to drive online profile-based optimizations for production systems.

HPCToolkit is a collection of multi-platform performance tools developed at Rice University that interpret the dcpi data.
- mpiP_2.5 is finally available on QA/QB as well as QSC. `module load mpiP_2.5` Usage is a matter of linking wrapper libraries ahead of your mpi libraries. The required library sequence is shown in computing.lanl.gov -> shared topics -> thirdparty -> mpiP mpiP provides statistics on MPI communication. Useful aspect is the association of time and count for each MPI function with the specific call sites in your program. Libraries are provided for R6 and R13 on QSC, for R5, R12, and R13 on QA/QB.
- tau_2.13.2 is now the default TAU module on both QSC and QA/QB. FORTRAN capability is improved. TAU-instrumented FORTRAN code may be linked with f90 rather than cxx, allowing fewer makefile changes. This version also provides improved FORTRAN auto-instrumentation. f95parse, which is based on flint, handles all of sage and nearly all of mcnp source code now. The original f90parse really only handles f77 code. For either C or FORTRAN there are options to auto-instrument subsets of the program routines. However limited, focused hand-instrumentation still seems to be the best use of TAU, regardless of language. tau_2.13.2 on QSC is configured for Alaska R13, on QA/QB for Alaska R12. Full manual is available in the installation directory.

Exercise

Practice Profiling With Pixie

- Recompile sweep-single.f using the `-g3` option.
- Then type `pixie -all sweep-single.exe` (assuming that was the name of the executable you prepared in the preceding step).
- Observe all the "stuff" produced in your directory.
- Then type `sweep-single.exe.pixie`
- Observe all the execution profile data produced in your directory (xxx.Counts).
- Then type
`prof -pixie -all -lines sweep-single.exe sweep-single.exe.Counts > pixout`
- Where is the majority of the time spent in this code? Was there any overhead associated with this pixie profiling.
- This completes the exercises and the course. Please be sure to fill out and return the course evaluation form.

Course Evaluation

Evaluation
Form

**Please complete the online evaluation form at
<http://trouble.lanl.gov/~hjl/eval.php> in the yellow network. You can click the blue box to go there.**

LA-UR 04-3787 (June, 2004)

Operated by the [University of California](#) for the [National Nuclear Security Administration](#),
of the US [Department of Energy](#). [Copyright © 2004 UC](#) | [Disclaimer/Privacy](#)



Contact Us : consult@lanl.gov

Last Modified: July, 2005

